

User Guide:

Zend Platform V.2.1.2 for i5/OS



By Zend Technologies, Inc.

Zend Platform User Guide Disclaimer

The information in this document is subject to change without notice and does not represent a commitment on the part of Zend Technologies Ltd. No part of this manual may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording, or information storage and retrieval systems, for any purpose other than the purchaser's personal use, without the written permission of Zend Technologies Ltd.

All trademarks mentioned in this document, belong to their respective owners.

© 1999-2006 Zend Technologies Ltd. All rights reserved.

Zend Platform User Guide for i5/OS issued August 2006.

DN: ZP-I5UG-170806-2.1.2-002

Table of Contents

Chapter 1 - Introduction to Zend Platform.....	1
About Zend Platform for i5/OS.....	2
<i>Navigation</i>	2
<i>Central Control Center</i>	2
Zend Platform Overview.....	3
Environments.....	4
Architecture	6
<i>Central Server.....</i>	7
<i>Nodes</i>	8
<i>Central-Node Communication</i>	10
<i>Single Point of Access GUI.....</i>	10
Chapter 2 - Configuring Zend Platform	11
Manage Clusters	11
Manage Servers	13
Manage Groups.....	14
Manage VHosts	14
Restrict Access to Virtual Hosts	15
Configure Event Triggers	16
<i>Why Configure Event Triggers.....</i>	18
<i>Filtering Event Triggers</i>	18
Define Event Triggers	20
Slow Script Execution Absolute	21
Slow Script Execution Relative	21
PHP Error	22
<i>Function Error.....</i>	22
Watched Functions File Event Types	23
Slow Function Execution	23
Excess Memory Usage (Absolute and Relative)	24
Database Error	25
Slow Query Execution.....	26
Inconsistent Output Size.....	26
Load Average	27
Custom Events	28
Define Event Actions	29
<i>Define Actions.....</i>	29
<i>Define Action Rules</i>	31
Configuring Performance.....	33
<i>Performance Tab</i>	33
File View	35
<i>Tree View</i>	36
<i>Tree View - Virtual Hosts List</i>	36
<i>File View</i>	38
<i>Define Caching Settings:</i>	38
<i>Modify File Settings</i>	39
<i>Fine Tune Caching Conditions</i>	39
<i>Define files to Blacklist</i>	40
Dynamic Content Caching	41
<i>Full Page Content Caching</i>	41
File Compression.....	45
Blacklists	47
Configuring the Zend Download Server (ZDS)	48
<i>Manual Mode</i>	48

<i>Transparent Mode</i>	49
<i>Testing the ZDS</i>	50
Clone Wizard	53
User Management.....	54
Adding and Editing Users	55
<i>Adding a User</i>	55
<i>Editing a User</i>	56
Adding and Editing Groups	57
<i>Add/Edit a Group</i>	57
User Settings	59
Chapter 3 - PHP Intelligence	61
The Problem Resolution Lifecycle	61
<i>Background</i>	61
<i>Implementing the Problem Resolution Lifecycle</i>	63
<i>Creating Events</i>	63
<i>Finding Events that Interest You</i>	64
System Health.....	66
<i>Event Reporting</i>	67
<i>Aggregation Groups</i>	67
Event List	68
Working with the Event List	69
Event Details	72
<i>Understanding Event Details</i>	72
<i>General Information</i>	73
<i>Event Occurrence Info</i>	74
<i>Zend Studio Diagnostics</i>	74
<i>Event Context</i>	75
<i>Show Source Code</i>	77
<i>Event Administration</i>	77
Controlling Information Displayed in an Event	78
Customizing Events.....	79
<i>Custom Events</i>	79
<i>Event Callbacks</i>	80
Aggregate Hints	82
Event Aggregation Rules	83
Database Maintenance	84
Chapter 4 - Performance	85
Overview	85
<i>Code Acceleration</i>	85
<i>Code Optimization</i>	86
<i>Dynamic Content Caching</i>	86
<i>File Compression</i>	86
<i>Download Server*</i>	86
Performance Lifecycle	86
Implementing the Performance Lifecycle	89
<i>Benchmarking - Site Analysis</i>	89
Analyze Site (Benchmark)	90
<i>Benchmark Web applications:</i>	90
<i>Test Results</i>	90
<i>Performance Test</i>	91
<i>Compression Test</i>	91
<i>Display Previous Test Results</i>	92
Event Trigger Settings and Analysis.....	93
<i>Calibrating Event Triggers for Performance Optimization</i>	93

Table Of Contents

<i>Investigating Performance Related Events</i>	93
Performance Optimization Tools.....	95
<i>When to Apply Optimization Tools</i>	95
Content Caching (Dynamic)	96
<i>Caching with the File View</i>	96
<i>Caching with the Performance Test Reports</i>	96
<i>Caching Alternatives</i>	97
Code Acceleration.....	98
<i>Accelerator Duplicate Functions Fix</i>	99
<i>Reset Accelerator</i>	99
Code Compression.....	99
<i>Setting Code Compression</i>	100
Zend Optimizer	100
Test Download	101
Viewing Download Test Results	102
Chapter 5 - Java Bridge	103
About Zend's Java Bridge Technology.....	104
<i>Zend Platform Node</i>	104
<i>J2EE Application Server</i>	105
<i>Added Value</i>	106
<i>Operating and Configuring Zend Platform's Java Bridge</i>	106
Java Bridge	107
<i>Java Status Page</i>	107
<i>Configuring the Java Bridge</i>	109
Usability Issues.....	110
<i>Chain Functions Call</i>	110
<i>Exceptions</i>	110
<i>Java Array/Hashtable Objects</i>	112
<i>Iterators</i>	112
Common Tasks	113
<i>Usage Scenarios</i>	113
<i>Activities</i>	113
Chapter 6 - APIs and Directives.....	119
Zend Platform APIs	119
<i>Accelerator Functions</i>	119
<i>Output Cache Functions</i>	119
<i>Monitor Functions</i>	121
<i>ZDS (Zend Download Server)</i>	122
Zend Platform Directives.....	123
<i>Accelerator Directives</i>	123
<i>Monitor Directives</i>	124
<i>Zend Monitor Event Types</i>	127
<i>GUI Directives</i>	127
<i>Collector Center Directives</i>	128
<i>Debugger Directives</i>	128
<i>ZDS Directives</i>	129
Chapter 7 - Tutorials.....	131
Integrating Existing and Legacy Applications	131
Calling an EJB on Websphere from PHP.....	135
Partial and Preemptive Page Caching	137
<i>Partial Page Caching APIs</i>	137
<i>Action Based Partial Page Caching</i>	144
Chapter 8 - Appendixes	147

Appendix A - Configuration Check List	148
Appendix B - Performance Lifecycle Check List	149
Appendix C - Event Aggregation Mechanism	150
Appendix D - Zend Platform Support	153
<i>Zend Platform Support</i>	153
<i>Zend Support Center</i>	153
Appendix E - zend.ini Configuration Settings	154
<i>Accelerator Directives</i>	154
<i>Monitor Directives</i>	155
<i>Debugger Directives</i>	157
<i>ZDS Directives</i>	158
Chapter 9 - Index	159

Preface

Zend Platform is an extremely diverse, runtime-environment management platform. As such, a greater understanding of the underlining concepts is required to fully benefit from the Zend Platform capabilities and features. This User Guide has been designed to reflect these concepts by providing a workflow driven description of Zend Platform's features.

The first chapter is an introduction describing the background and architectural design of Zend Platform along with who should read this guide and how to maximize the benefits of deploying Zend Platform in your environment.

The chapters that follow are each dedicated to a different facet of working with Zend Platform starting with initial configuration actions. The next chapters highlight the different Zend Platform features and how then can be employed in a working environment.

The functionality of each screen is further elaborated in the online Help that is included with Zend Platform, to provide detailed instructions on how to work with Zend Platform.

Workflow related processes also addressed in the last chapter dedicated to tutorials that include detailed descriptions and instruction on how to perform each process.

Audience

Zend Platform is responsible for providing solutions to the challenges faced by different stakeholders in the organization, therefore it is recommended that Managers, System Administrators and Developers read this manual:

Managers will learn how to utilize this solution to support PHP lifecycle management by streamlining the PHP application lifecycle across development and production.

This kind of development method is responsible for shortening release cycles by integrating the working environment. Managers can benefit from knowing how to utilize Zend Platform with their testing staff to obtain a means for detecting and pinpointing run-time problems throughout the production lifecycle while having the safety of knowing that when run-time problems are located, testing staff will have a complete audit trail to help resolve the issue.

From the usability aspect, delivering a product on time is only part of the equation. Find out how to improve user experience and increase performance, up-time and customer satisfaction through using Zend Platform.

From the PHP aspect, know why Zend Platform considerably improves PHP execution while Maintaining and synchronizing PHP configurations.

From an organizational standpoint, find out how to leverage existing investments in Java applications as well as reducing costs on hardware.

System Administrators will be able to understand how to detect problems with PHP Intelligence to pinpoint run-time problems with detailed information. Learn how to control configuration by synchronizing configuration of .ini files accurately across PHP servers and improve performance by getting more out of each PHP server (less servers doing the same job)

Developers can learn how to integrate with Zend Studio to improve quality by quickly identifying problems and reducing the testing cycle. They can also benefit from the PHP/Java Integration Bridge to reuse code by utilizing existing code.

Chapter 1 - Introduction to Zend Platform

IN THIS CHAPTER...

ZEND PLATFORM OVERVIEW

ENVIRONMENTS

ARCHITECTURE

CENTRAL SERVER

NODES

CENTRAL-NODE COMMUNICATION

SINGLE POINT OF ACCESS GUI

Zend Platform is a complete runtime environment for managing and maintaining mission critical and enterprise PHP applications from a single, centralized location.

This environment consists of cluster management; performance management, monitoring, detection and recovery; and Java integration.

Zend Platform improves both the end user experience and IT productivity by combining cluster and performance management; automated monitoring and detection capabilities; and powerful Java Integration capabilities into one integrated environment.

Zend Platform provides the PHP-enabled enterprise with the following benefits:

- The ability to manage every aspect of PHP from a single, Web-based interface
- The ability to quickly drill-down to critical issues to resolve and optimize
- The ability to create user defined thresholds and error values
- The ability to configure servers from a remote management station and to perform controls at a click of a button
- The ability to clone servers: one-to-one and one-to-many
- The ability to monitor performance improvement with Code Acceleration, Content Caching, File Compression and the Download Server
- The ability to integrate with Java system elements over Platform's fully implemented PHP/Java Bridge.

About Zend Platform for i5/OS

Zend Platform is the only robust PHP production environment that ensures your applications run smoothly at all times.

Designed for IT personnel and businesses that require industrial-strength applications in highly reliable production environments, Zend Platform offers high performance and scalability to provide your customers with the best possible Web experience and response time.

Zend Platform uniquely guarantees application up-time and reliability through enhanced PHP monitoring and immediate problem resolution that removes the troubleshooting guesswork out of the equation and replaces it with peace-of-mind.

You spent time and money developing your state-of-the-art PHP application, now it's time to ensure it's up and running.

Navigation

Zend Platform is a browser-based application. The general layout of functionality is in a tabbed view where each tab represents a unique functionality.

- Zend Central - central management functionality
- PHP Intelligence - monitoring and event generation capabilities
- Performance - performance enhancement tools
- [Java Bridge](#) - incorporate a Java environment to enrich your applications

Central Control Center

Zend Platform is designed as to handle clusters and not just standalone servers. As such users can navigate freely between the central server and nodes.

Users stay on the central server as long as they are not prompted to select a server. As soon as a server is selected all subsequent actions and settings will be done on the selected server only.

A status bar showing the date, time and login name will also display the name of the server the user is currently on.

When no server name appears you are on the central server.

Zend Platform Overview

Zend Platform is a central management solution and run-time environment for:

- **Configuration Management** - Platform's architecture provides full control of the PHP application platform, including performance management settings, event thresholds, etc. allowing administrators to set up groups of multiple identical servers via:
 - Remote server configuration.
 - Clone configurations or parts of configurations from one server to another or from one server to an entire group of servers.
- **Performance Management*** - Platform is equipped with four management modules for tracking and improving speed and responsiveness of Web applications. These include: Code Acceleration, Dynamic Content Caching, File Compression and the Zend Download Server.
- **PHP Intelligence** - Platform features new technology that detects and recovers crashes, whether they occur in PHP itself, the database software, or your own application. The integrated suite of monitoring, detection and recovery features allows users to drill down to critical issues and optimizations quickly and easily.
- **PHP/Java Integration** - The Platform PHP/Java Bridge module provides PHP centric companies with a well-rounded environment making sure that the organization benefits from the "best of both worlds". Be it, existing investments in J2EE application servers that require this solution, or to provide a means for organizations - if they choose, to bridge language limitations by use of Java applications. The Java Bridge is not limited to interactions strictly with J2EE and legacy systems, the Platform PHP/Java Bridge also provides the ability to interact with plain Java objects.

Environments

A typical environment for running any Web application consists of three basic components: Web servers for running the Web application, a load-balancer to handle traffic and a firewall to protect from unauthorized entry into the hosting network.

Zend Platform, once introduced to this kind of an environment becomes a control environment for web server activity.

In an environment where a single web server manages activity, Zend Platform resides on the web server to provide system health and analysis information.

Moreover, environments that include several web servers, be they a cluster servicing a single Web application or a collection of clusters servicing different Web applications, Zend Platform serves as a single control center for system health information, cluster management and runtime process optimization.

The Zend Platform system diagram below, demonstrates where Zend Platform components typically reside in the PHP- enabled enterprise.

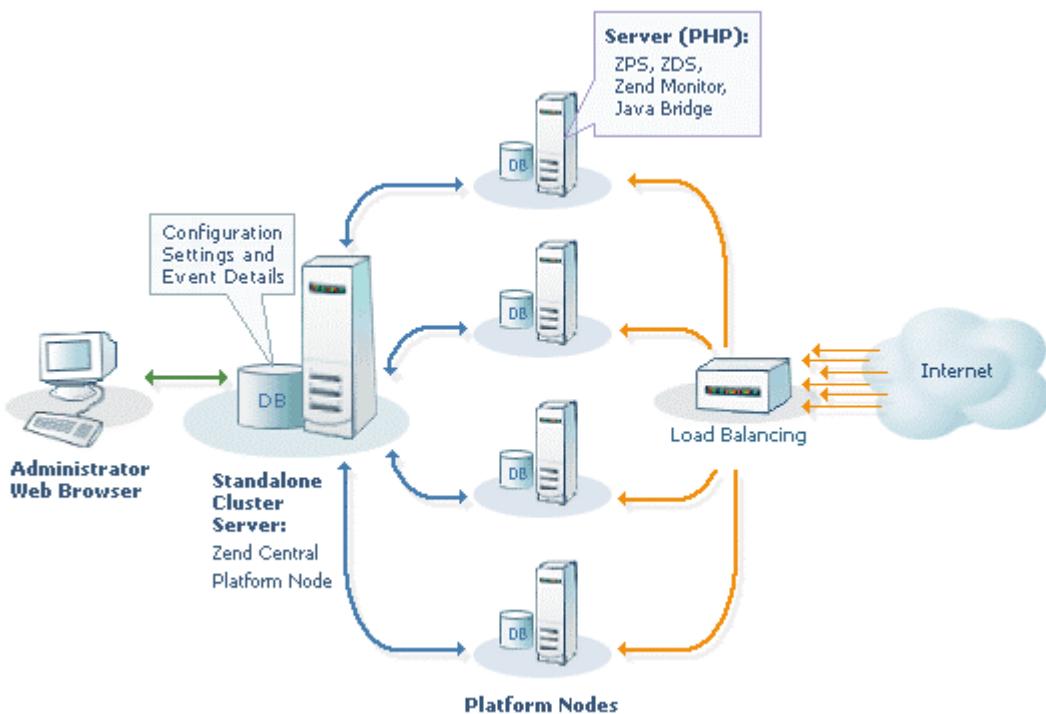


Figure: 1 - Zend Platform System Diagram

The system diagram illustrates the following points:

- Zend Platform's Standalone Cluster Server is installed on a Web server.
- The System Administrator controls all Platform Central functions. Providing the ability to work with Platform from a single workstation using a standard Web Browser.
- Nodes host resident PHP-based services that fill requests from the Web.
- Load Balancing directs requests to available servers in the web farm.

Note:

Platform Server and the Platform nodes are separate entities, therefore it is important that firewall and security devices be configured to allow communication between the nodes and the Platform Server. Identify which ports are being used and if necessary, open these ports on your firewall.

Architecture

Zend Platform is a complete environment that provides rich functionality by interacting with the existing PHP in a simple and generic way. Zend Platform is a non-intrusive extension to an existing environment with minimal overhead that helps obtain enhanced performance and reliability.

Zend Platform extends the Zend Engine with the organization's execution environment, providing the platform on which Web services, business to commerce applications, content-management, Intranet and business-to-business applications are based.

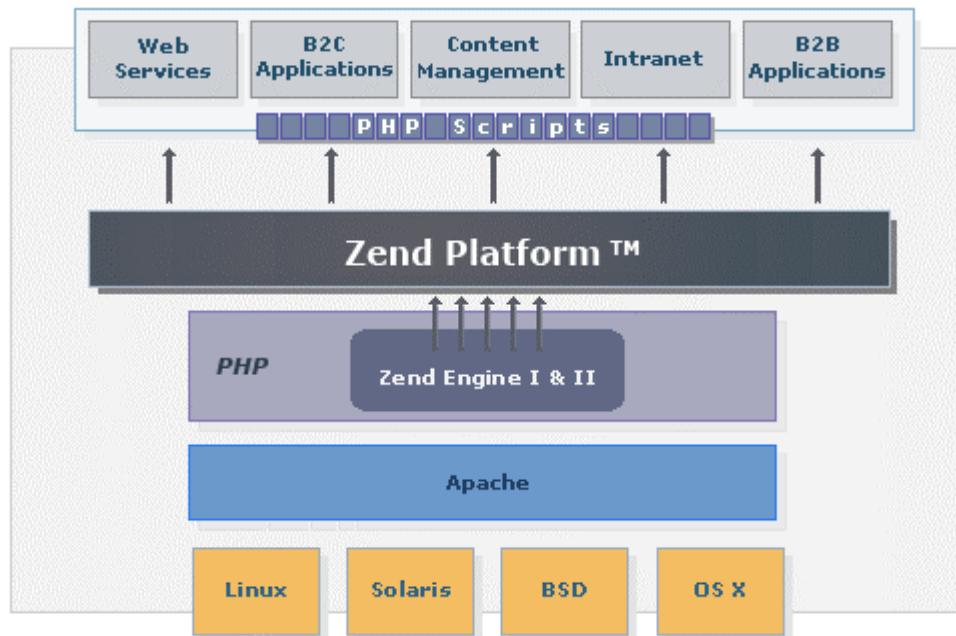


Figure: 2 - Zend Platform and the PHP-enabled Enterprise

Zend Platform consists of two deployed components the Central Server (consists of a Server + Node) and the Node component.

Zend's Central Server is a central management component for governing node configurations and script performance information. The Central Server can be deployed as a standalone Zend Platform environment for a single server and for this reason, contains fully functioning node components. However, the prominent application for Zend Platform is multiple server/cluster based environments.

Zend Central provides a single point of access and control for multiple nodes.

Nodes are web servers that run with Apache and service a PHP application. The Zend Platform components are installed on the node to report script, database and system activity to the Central Server. Each node installation also includes a debugger that is integrated with Zend Studio extended code management features such as profiling, debugging and correcting code directly on a node.

In essence, similar components are installed on the Central Server and the Nodes since the Central Server also performs as a node. However, the Central Server and the Node Components employ different modules for their overall activity.

Central Server

The Central Server provides the necessary functionality for handling event information, node management and performance monitoring.

No matter how many nodes are registered in the cluster, from the users point of view Zend Central provides an efficient and useful single point of entrance. Zend Central resides on the Central Server and is in charge of displaying the GUI for the Central Server and Node configuration. Zend Central is the main communication component for collecting, storing, configuring and receiving information from the nodes.

Communication is carried out via regular TCP/IP communication and event information is stored in a dedicated database. Zend Central governs the PHP application performance and monitoring features including configurations for nodes, PHP and event collection.

The following illustration is a representation of Zend Platform Server components:

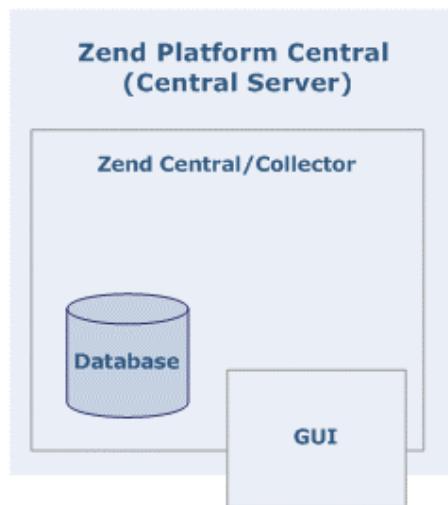


Figure: 3 - Zend Platform Server Components

The Central Server is dedicated to controlling the nodes, to this server the Zend Platform Central server is installed. The installation includes three main components:

1. Zend Central that includes information collection and functionality: Zend Performance, PHP Intelligence and the Java Bridge.
2. The Database is the main repository for event information collected from all registered nodes.
3. One of the main components of Zend Central is the Collector. Through this component information from nodes in the cluster is collected aggregated and displayed in the Zend Platform PHP Intelligence module. The collector collects and aggregates information according to rule configurations that can be applied to a single server or to several servers.

Nodes

The Nodes are the web servers that run PHP. Nodes are the individual servers that service a Web application and a collection of nodes also referred to as a cluster. The central server governs clusters.

The following components need to be on each Node:

- Basic:
 - A supported operating system (Linux, Unix etc)
 - A Supported Web Server (Apache)
- PHP:
 - PHP version 4 or 5
- Zend Products
 - Zend Platform
 - Zend Download Server
 - Zend Java Bridge
 - Zend Optimizer
 - Zend Debugger

Nodes have to be registered with the Central Server in order to enable communication between the Node and the Central Server. There are two ways to register a Node to the Central Server: the installation process or through manually registering the Node .

Zend Platform Nodes consist of several components that report information to the Central Server and provide debug capabilities for PHP scripts residing on a node.

- A Collector Component for transferring event information to the central
- Debug Infrastructure for debugging live pages directly from a node (This option is supported by Zend Studio and Zend Core)

The following illustration is a representation of Zend Platform Node components:

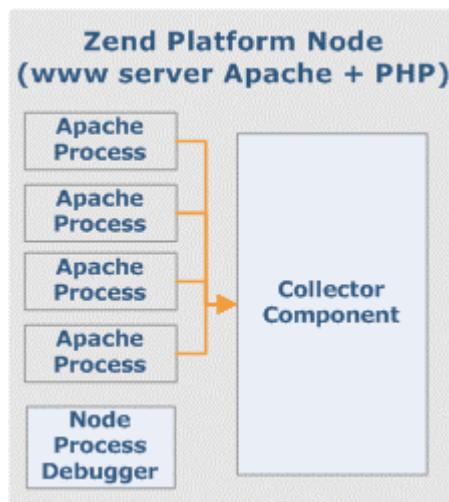


Figure: 4 - Zend Platform Node Components

The Collector component listens to the running processes and collects event information (For more on Events go to "Configuring Events,"), to be reported to the Central Server over a regular TCP/IP connection using SSL. However, only if the node has the appropriate certificate indicating that it is part of the cluster will the Central Server agree to receive event information from a node's collector.

The type of information the Collector listens to and collects is event information that is determined by Event Rules that are configured on the Central Server. Event information is sent to the Central server where it is aggregated according to event type (more about event aggregation can be found in Appendix D - "Event Aggregation Mechanism"). Different Rules can be applied to different nodes in a cluster environment or specific settings can be applied to more than one node in a cluster (by using the Clone Server feature).

The Debugger Infrastructure is enabled via the Zend Studio/Zend Core Communication Tunnel that is geared to work in development and production environments. With the appropriate configuration the Debugger Infrastructure can work through Firewalls or NAT devices that may be positioned between the Node and Zend Studio. The Debugger Infrastructure provides full lifecycle support for editing debugging, profiling and deploying code by enabling to view and edit Event source code in the Zend Studio development environment. This provides Zend Studio users with access the remote debugger via the same communication tunnel that routes full-duplex traffic over HTTP. The Debugger Infrastructure utilizes the Communication Tunnel, ensuring that multiple servers can be debugged through the same Communication Tunnel at once.

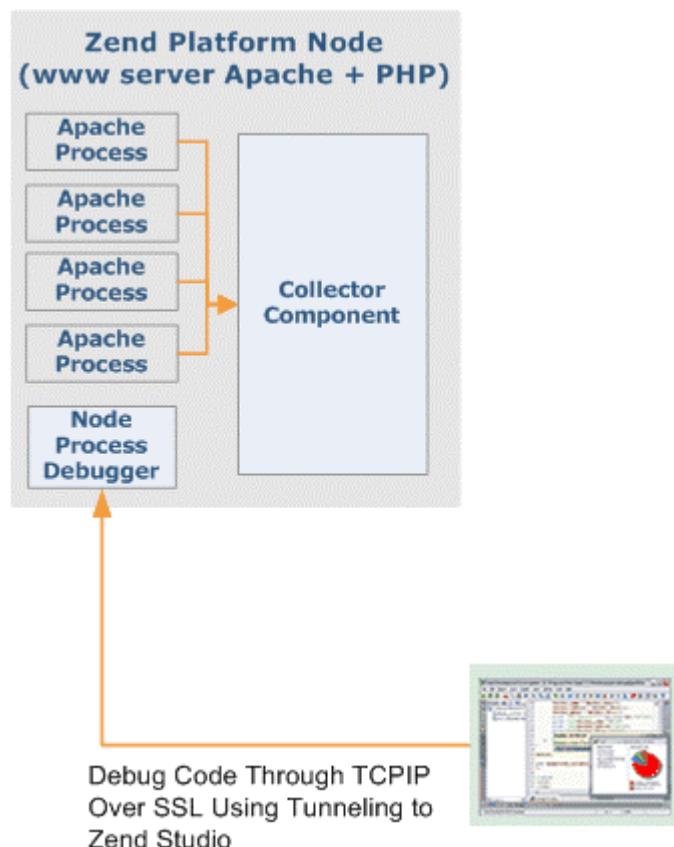


Figure: 5 - Communication with Zend Studio

Central-Node Communication

Traffic between the Central Server and Node clusters mostly occurs from the nodes to the central server with the nodes reporting event information through the collector component to Zend Central.

However, Zend Platform has a Server Status feature that periodically checks the availability of each Node in the cluster and provides up to date information regarding the components installed on the nodes.

The following diagram illustrates the communication between the Central Server and Nodes in a Cluster:

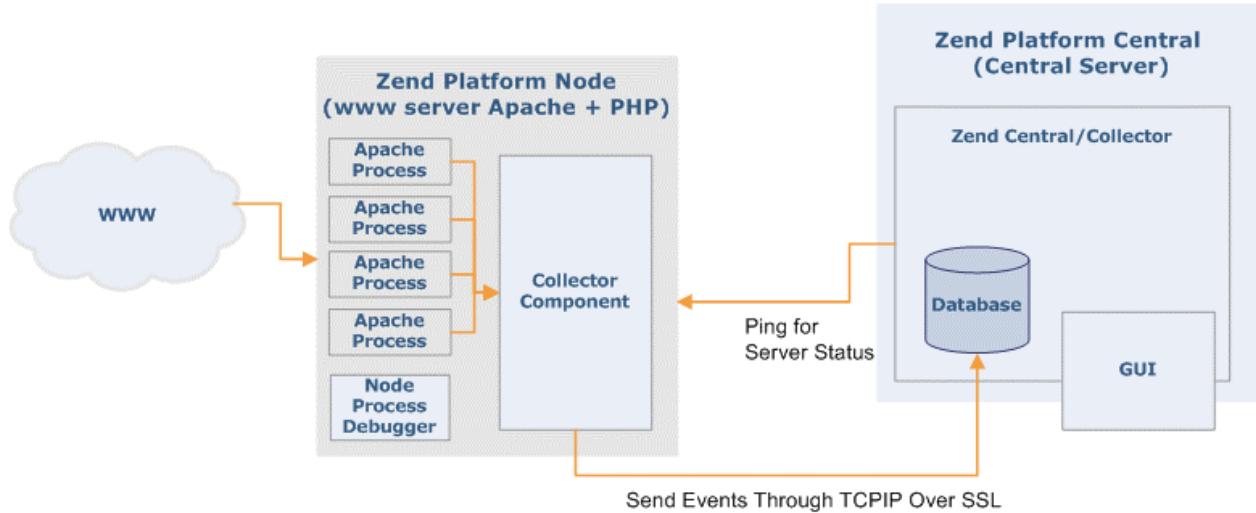


Figure: 6 - Central Node Communication

Single Point of Access GUI

Zend Platform's sophisticated architecture enables to use the Central Server as a single point of access for node availability and configuration, enabling to configure node settings and behavior from the Central Server itself. This connectivity is achieved by the addition of GUI components on the Nodes as well as on the Central Server in the installation process. In this process the Central Server's URL is specified to the Nodes as a central control unit and from that point onwards, access and read write permissions to nodes, can be established from the Central Server.

Chapter 2 - Configuring Zend Platform

INSIDE CONFIGURING ZEND PLATFORM

MANAGING CLUSTERS

CONFIGURE EVENT TRIGGERS

DEFINE EVENT TRIGGERS TABLE

CONFIGURE EVENT ACTIONS RULES

CONFIGURE PERFORMANCE

CONFIGURING THE ZDS (ZEND DOWNLOAD SERVER)

CONFIGURING PHP SETTINGS

CLONE SETTINGS

CONFIGURING USERS AND USER PERMISSIONS (ROLES)

The Zend Platform installation, produces an out-of-the-box fully functioning version of Zend Platform. This installation includes basic default settings for monitoring events and code acceleration. At this stage Zend Platform already generates events and improves code generation. However, to benefit from Full-Power Cluster Management, Audit Trails, and much more, it is necessary to tune Zend Platform's performance settings to suit your individual work environment.

In this chapter, each configuration task is detailed in a chronological order beginning from the initial configuration tasks to configurations that may rely on other settings.

The configuration actions addressed in this chapter are listed below:

- **Manage Clusters** - add the servers that you want Zend Platform to control. Each server should be added and then grouped to create a cluster environment to be treated as a single entity in terms of event collection.
- **Configure Event Triggers** - customize the Event Triggers to suit your working environment. The Zend Platform installation comes ready with default configurations; however, it is recommended that a person with an understanding of the environments settings and performance standards, configure Event Triggers accordingly.
- **Configure Event Actions** - once Event Triggers are configured the next logical step is to determine the actions and action rules that can be applied to Events generated according to Event Triggers.
- **Configure Performance** - adjusting performance requirements is a way to benefit from the Zend Platform advance performance features.
- **Clone Settings** - once all initial settings have been configured, the Clone Settings feature can be used to apply settings to other nodes in one single step.
- **Configuring Users and User Permissions (Roles)** - granting different levels of permissions to different users provides a means for controlling actions performed in the environment and for enforcing work procedures. This is the last step to customizing Zend Platform to your working environment.

Manage Clusters

Zend Platform manages clusters to make them available and manageable from a single

location – the Central Server. Clusters are identified as a single unit for monitoring and management purposes. However, each node in the cluster can be individually accessed from the Central Server.

(Events originating from different servers are aggregated; however, they include an identifier for each node on which the Event occurred).

Servers are added to Zend Platform via the installation process. Once a server is added, server settings can be applied.

The cluster setting options available from: Zend Central | Console | Manage Cluster are:

- [Manage Servers](#) - Configure, delete and define servers.
- [Manage Groups](#) - Group servers together for event reporting and configuration purposes.
- [Manage VHosts \(Virtual Hosts\)](#) - Manually delete and define Virtual Hosts

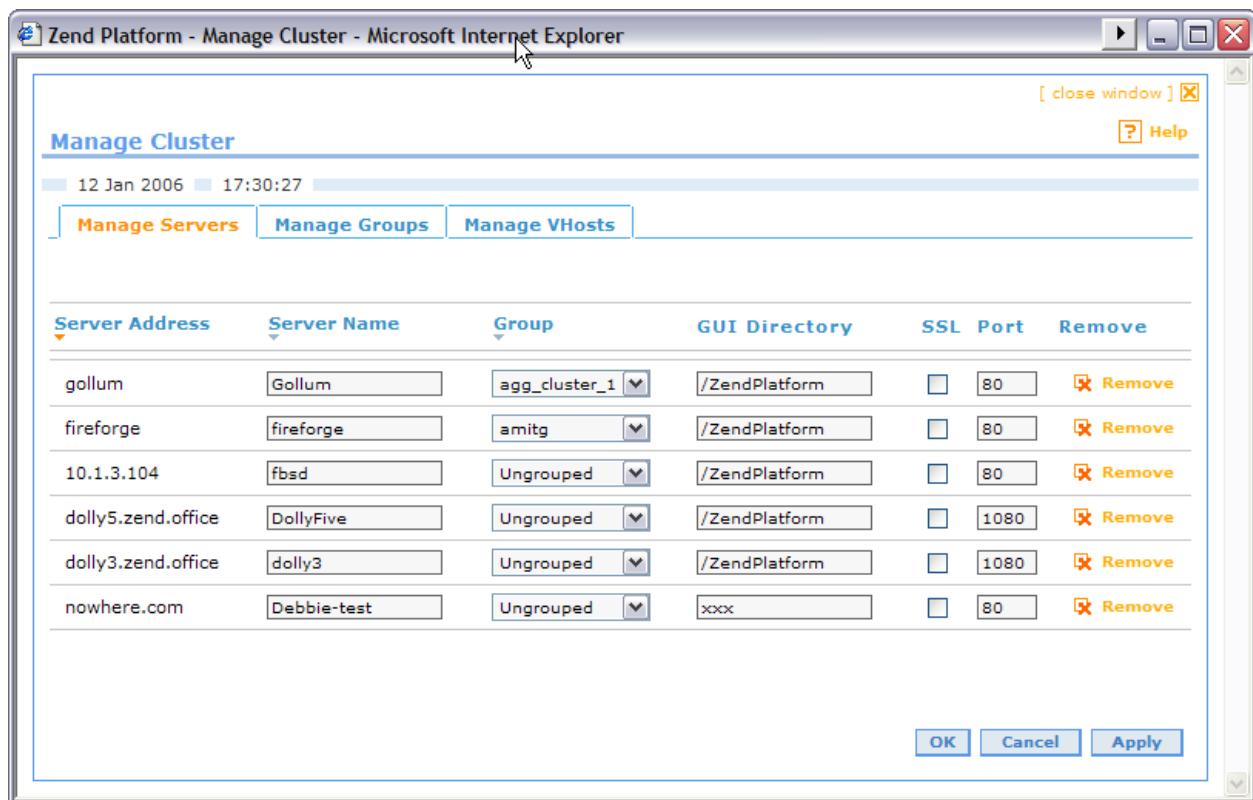


Figure: 7 - Manage Clusters Dialog

Once servers have been configured and grouped, Event Triggers can be defined.

Manage Servers

The Manage Servers tab provides options for configuring and defining settings for servers that were added to Zend Platform using Zend Platforms Node installation script.

Only servers that have been installed with the Zend Platform Node installation will appear in the Manage Server tab.

To add a server to Zend Platform so that it appears in the Manage Servers tab:

1. Run the Zend Platform Node Installation Process (Please see the Zend Platform Installation Guide for details on Node installation).
2. Zend Platform automatically identifies registered servers and displays them in the Manage Servers tab.
3. The installation script sets the Server Name.

Now the new server's settings can be defined.

Server settings are defined from: Zend Central | Console | Configuration and Management Tools | Manage Cluster.

The Server management settings are as follows:

- **Server Address** - The actual hosts address (not editable).
- **Server Name** - The server's name for identification and all references to the server from Zend Platform.
- **Group** - Designate a server to an existing group (new groups are added to the list from the Manage Groups tab).
- **GUI Directory** - States the location of the server's GUI Installation.
- **SSL** - Check the box if the server uses SSL.
- **Port** - Specifies the port with which the specific server works.
- **Remove** - Removes the server from the database (unregistered) and deletes all events related to the removed server.

These settings should only be changed if changes that may affect these settings, occurred since the node installation.

Note:

To add a removed server you can re-register by going to the server and running: ...<install dir>/bin/register_node.sh (currently not applicable for Windows Operating Systems).

Manage Groups

The Manage Groups tab provides options grouping servers together for event reporting and configuration purposes.

Groups are created for three distinct purposes:

- To aggregate Events across nodes (only if the nodes are running the same Web application).
- To enable configurations to be automatically applied to other servers belonging to the same group (using the Clone Configurations feature).
- To facilitate handling and managing groups of servers.

Note:

Groups should only be aggregated when the PHP application on all servers in the group is identical.

To create a new group:

1. Give the Group a name in the "Add a new group field" and press Add.
A new group will be added to the list below.
2. If you want to aggregate all events that occur on the servers associated with the specific group select the Aggregated option.

Manage VHosts

The Manage VHosts tab provides a way to manually define Virtual Hosts.

In general, virtual hosts are automatically added based on Event activity. However, Virtual Hosts only appear in the lists after an event is generated for a specific virtual host.

To ensure that all Virtual Hosts can be visible, an additional option has been added to manually add Virtual Hosts. This option allows users to create the actual virtual host list for any given server.

Virtual hosts can be added or deleted from this tab.

Virtual hosts are added per server. However, virtual hosts can be deleted in one of two ways:

1. **Per virtual host name** - do not select a specific server name before adding or deleting the virtual host.
2. **Per virtual host for a specific server** - select a specific server name before adding or deleting the virtual host.

When deleting a virtual host all the events related to that virtual host are permanently deleted from the event database.

Restrict Access to Virtual Hosts

Defining Virtual Hosts provides a way to prevent certain users from gaining access to information regarding certain Virtual Hosts. This provides an additional level of authorization that is more precise than granting permissions per server.

To restrict user permissions by virtual host:

1. Go to Zend Central | Console | User Management.
2. Select the server on which you want to restrict permissions and click the edit button. The Edit User Wizard opens (see [Adding and Editing Users](#) for complete instructions on the Edit User Wizard).
3. In step 2 make sure the check box next to "No Server Restriction" is left unchecked.
4. Select the check box next to the virtual hosts that should be granted for access. This will grant access to the selected virtual hosts only
5. Press finish to close the Edit User Wizard and return to the User Management Screen.

Configure Event Triggers

Event triggers can be customized to suit your working environment. Zend Platform comes ready with default configurations. However, a person with an understanding of the environment's settings and performance standards should construct the Event Triggers to suit each unique environment. Event Triggers define the conditions under which an event, will be captured by the monitoring system (PHP Intelligence).

To Configure Event Triggers, go to PHP Intelligence | Event Triggers or use the Shortcut from the Zend Central Console.

Users are prompted to select a node before entering the Event configuration screen as all configurations are made to a selected node. The top bars of screens display the name of the node, no name means the user is working directly on the Central Server.

For example, the image below displays the following text: Server name "Fireforge". This means that the user is no longer working on the Central Server but working directly on the node (in our case a node who's alias is Fireforge).

The screenshot shows the 'Define Event Triggers' page in Zend Platform. At the top, there's a navigation bar with links for Zend Central, PHP Intelligence, Performance, Studio Server, Session Clustering, Java Bridge, Logout, and Zend. Below the navigation is a toolbar with buttons for Save Rules, Clone Event Triggers, Refresh, and Help.

The main area is titled 'Define Event Triggers' and displays a table of event triggers. The table has three columns: 'Event Type', 'Active', and 'Rules'. The 'Event Type' column lists various monitoring events like Slow Script Execution, PHP Error, Function Error, Slow Function Execution, Excess Memory Usage, Database error, Slow Query Execution, Inconsistent Output Size, Load Average, and Custom Event. The 'Active' column contains checkboxes, many of which are checked. The 'Rules' column contains configuration details for each event type, such as runtime thresholds (e.g., 'Script runtime exceeds 500 msec') and additional rules (e.g., 'Suppress in case the load average is above 3').

Figure: 8 - Define Event Triggers

The Event Triggers screen is used for defining and modifying Event Triggers to monitor events on a specific node. The table is used to define the conditions under which an event will be captured by the monitoring system.

The possible actions on this screen are:

- Configure Event Triggers for a specific server.
- View Event Triggers currently defined for the node.
- Filter the view of events displayed in the "Define Event Triggers" table.

- Clone Events

To configure Event Triggers:

1. Click "Event Triggers" in the "Configuration and Management Tools" list of shortcuts. The Select Server to Configure screen opens.
2. Select a server from the "Server Tree."
3. Click Select to open the "Event Triggers" screen for the selected server.

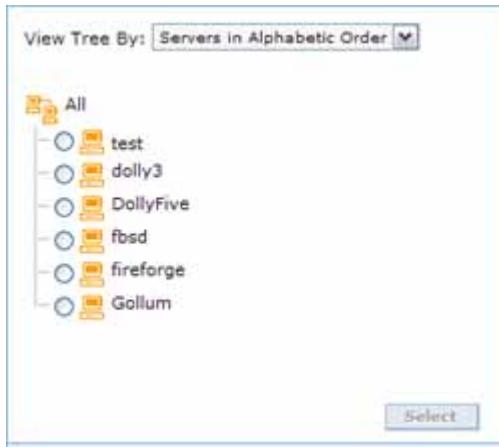


Figure: 9 - Select Server to Configure

Notes:

To configure multiple servers or a server group with the same Event Triggers: (1) configure a selected server and (2) then use the Quick Clone button to propagate settings from that server to other server nodes.

Event Types are the specific events configured by administrators for monitoring a server/node. The Define Event Triggers procedure is used to define the conditions under which a server will generate an Event of a specific severity when an event (of the type) occurs. The Event Types supported in the current version of Zend Platform are described below.

Why Configure Event Triggers

Event Triggers are an essential tool for pinpointing bottlenecks in Web applications. Events not only indicate that one of the thresholds was breached they also collect information relevant to the occurrence to provide a full audit-trail for diagnostics.

In terms of the end product, these thresholds can be directly translated into performance issues the end user may encounter. Therefore, the more Events resolved the better the application will run.

By using Event Triggers, scripts can be monitored to identify with precision the number of milliseconds or percentage it takes to execute a script. This identification is based on parameters that you can determine as acceptable performance thresholds.

Filtering Event Triggers

Zend Platform is equipped with 12 types of Events for monitoring performance and script execution. The default Event Trigger display is a non-filtered view that shows all the available Alerts. A filter is provided to allow displaying a selection of events by type.

To filter events:

1. Press (Filter By) to expand the filter list.
2. Use the two drop-down fields to select the Events to display by:
 - **Events From** – The area where the event originated (script, database, web server, etc.)
 - **Event Types** – Filter view to display Events according to their Event Type (The selection changes according to the area chosen in the “Events From” field).
3. Press (Go) to filter the view.

Define Event Triggers

The fields that make up the Define Event Triggers table can be understood as follows:

- **Event Type** - The type of event that, under the rules defined, will produce an alert in the monitoring system.
- **Active** - When enabled for a specific event, Zend Monitor (node) will report alerts when they occur (This gives the user the right to disable an event for a particular server).
- **Rules** - Defines the conditions under which an event will produce a report. For example, (Red) Script Runtime Exceeds 500 Seconds means that the system will generate a critical (red) event—for Slow Script Execution (Absolute) type events—if the condition (> 500 sec.) is met.

Note:

The user defines the thresholds for both the moderate and severe events. Some events have only one level of severity (like function error).

To define whether or not Zend Monitor will report a specific event, enable/disable the event in the Active column of the Define Event Triggers table.

To save the changes to Event Trigger definitions, click Save Rules.

The changes will then be updated in the database.

Each event type has its own advantages and characteristics. Press the link to view all the different Alert Types, their descriptions and recommended usage. Choosing and Defining Event Triggers

Note:

Events marked as “Performance Monitoring Events” have a special role in optimizing web application performance.

Slow Script Execution Absolute

This is a performance monitoring event

Absolute Slow Script Execution is used to generate an event when executing a script exceeds defined limits. This function is used to maintain performance standards.

Default parameters are 500 msec for moderate, 2000 msec for severe alerts.

Additional Rules:

- Suppress in case a "Slow Function Execution" event occurs. Selecting this option ignores "Slow Script Execution" events that are caused by a slow function. This is to prevent double reporting, as these events will be reported as "Slow Function Execution" events.
- Suppress in case the load average is above X - Selecting this option ignores events that occur when the average number of active processes waiting for CPU time is above x active processes (3 active processes is the default value).

Note:

These additional rules are applied to the Absolute and Relative Slow Script Execution event types.

Slow Script Execution Relative

This is a performance monitoring event

Relative Slow Script Execution is used to generate an event when script execution is lower or higher than the average script execution time. Parameters should be set to a certain percentage for moderate and severe alerts.

The default values for this event type are set to 0. To generate events, configure these settings to a value that suits required script run-time.

Additional Rules:

- Suppress in case a "Slow Function Execution" event occurs. Selecting this option ignores "Slow Script Execution" events that are caused by a slow function. This is to prevent double reporting, as these events will be reported as "Slow Function Execution" events.
- Suppress in case the load average is above x - Selecting this option ignores events that occur when the average number of active processes waiting for CPU time is above x active processes (3 active processes is the default value).

Note:

These additional rules are defined in the Absolute Slow Script execution cell.

Relative Events: event definitions are based on relative values i.e. percentage. Relative values are set according to warm-up settings, default value of 500 requests. If necessary, the default value can be modified manually by changing the zend_monitor.warmup_requests directive in the zend.ini.

PHP Error

PHP Errors are used to identify all types of PHP errors such as:

- Hard errors that cause stops in page execution.
- Warnings that interrupt the end user experience.
- Notices that could lead to larger problems.

This type of event is useful in QA processes to identify problems that may have slipped through the cracks during production.

Description:

Used to generate severe or moderate events on selected PHP errors, when they occur, and identify real-time failures for given users.

To select a PHP Error Level, scroll through the selection and use CTRL for multiple selections. The trigger types lists are the same; therefore severe event selection takes priority over moderate event selection.

Additional Rules:

Event reporting for PHP errors can be changed by setting error reporting to 0 or using the silence operator @.

There are three options for activating Additional Rules:

1. Always Report Errors - Ignore the error-reporting setting and the silence operator and report all PHP errors.
2. Report errors that match the error-reporting criteria - Ignore all PHP errors that were silenced using either the error-reporting setting or the silence operator.
3. Report any errors that were not silenced with the @ operator - Ignore the error-reporting setting and only ignore errors that were silenced with the silence operator.

Function Error

Function Errors are returned from functions and therefore offer more specific information about the root of the error that does not always arise from PHP errors.

This type of Event is usually used in QA and Production for identifying run-time events, as opposed to PHP errors that identify code-related/syntactical events.

Function Errors can prove to be invaluable to an organization as they provide a different perspective on problems (view the outside problems through the eyes of PHP). Despite the fact that the code may be running okay, this Event indicates what other outside problems (i.e. network, database, web services, file system etc.) you may have, based on the PHP function's behavior. Issues like these used to be difficult to reproduce however with the complete audit trail and full problem context, Function Errors can be easily reproduced to a level of accuracy that mirrors the actual time of occurrence.

Description:

Generate a severe event when an error in one of the specified PHP Functions (built-in or user-defined) fails (returns a FALSE value).

To add a function, enter the name into the + field and press Add (+).

There are three ways to monitor PHP functions:

1. Specify the function name, object methods can also be used (for example, bar::foo).

2. Use wild cards (*) to specify a range of function names for example mysql_* will select all functions beginning with mysql_.
3. Specify the full path to a file containing a list of functions, each in a new line.

Note:

Database related functions are directed and reported as Database Errors (see the "Database Error" event type).

Watched Functions File Event Types

The Watched Functions file can add Function Error and Slow Function Execution event types (PHP Intelligence | Configure Event Triggers) by entering a function in the field and pressing Add or specifying the full path to a file containing a list of functions.

When the Watched Functions file is applied to the "Function Error" Event Type, the functions included in the file will be monitored and an Event Details screen will be generated.

Slow Function Execution

This is a performance monitoring event

Slow Function Execution events are used to identify bottlenecks within functions providing a more granular approach than finding bottlenecks in pages.

This type of event is useful in the production process for pinpointing performance bottlenecks by watching functions that the user specifies.

Slow Function Execution events provide a different perspective on problems (view outside problems through the eyes of PHP). Despite the fact that the code may be running okay, this Event indicates what other outside problems (i.e. network, database, web services, file system etc.) you may have, based on the PHP function's behavior. This Event is also useful for catching pure PHP functions that are performing slowly.

Description: Generates an event when function execution exceeds the setting defined in the rule. The default values are, 500 msec for moderate, 1000 msec for severe alerts. This applies to the functions selected in the additional rules section.

Additional Rules:

Generate events for specified PHP functions (built-in or user-defined).

There are three ways to add functions to be monitored:

1. Specify the function name, object methods can also be used (for example, bar::foo).
2. Use wild cards (*) to specify a range of function names for example mysql_* will select all functions beginning with mysql_.
3. Specify the full path to a file containing a list of functions, each in a new line.

Note:

Database related functions are directed and reported as Slow Query Execution (see the "Slow Query Execution" event type).

When the Watched Functions file is applied to the "Slow Function Execution" Event Type, the functions included in the file will be monitored and an Event Report will be generated when the function execution exceeds the values defined to trigger a moderate or severe event.

Excess Memory Usage (Absolute and Relative)

This is a performance monitoring event

(Absolute – a customer configured hard number; Relative – a customer configured percentage)

Excess Memory Usage events are used to identify when scripts are using excess memory that can hinder the application's ability to perform.

This event is used mainly used in Production but QA can also benefit from monitoring by KB or percentage of memory used by a script to execute.

Description:

- **Excess Memory Usage (Absolute)** - Generates an event when memory for PHP script execution uses more than a set amount of KB for moderate events and severe events.
- **Excess Memory Usage (Relative)** - Generates an event when memory use for PHP script execution is above or below average, a set percent for moderate and severe events.

Note:

Both Event Types are only active if the PHP is compiled with memory limit. (Compile the PHP, with the configure switch "--enable-memory-limit".

The default values for both of these event types are set to 0. To generate events, configure these settings to a value that suits required memory usage.

Relative Events:

Event definitions are based on relative values i.e. percentage. Relative values are set according to warm-up settings, default value of 500 requests. If necessary, the default value can be modified manually by changing the zend_monitor.warmup_requests directive in the zend.ini.

Database Error

Database Error Events are used to report function errors to databases such as:

- Connection errors
- Database selection errors
- General database function errors

These events do not require any additional configurations to the database. They can be used in production to delineate between a PHP problem and a database problem.

Database Errors can prove to be invaluable to an organization as they provide insight into the Database reliability along with a different perspective on problems (view outside problems through the eyes of PHP). Issues like these used to be difficult to reproduce however with the complete audit trail and full problem context, Database Errors can be easily reproduced to a level of accuracy that mirrors the actual time of occurrence.

Description:

Generates an event whenever a database-related function fails. This event is directly associated to the "Function Error" event and is activated and defined in correlation with this event type.

Database functions that should be reported are defined (or deleted) from the "Function Error" functions list.

Note:

To view supported databases, see the database related function prefixes listed in:
<install_dir>/lib/db_functions.txt or in windows <install_dir>\lib\db_functions.txt.

Slow Query Execution

Slow Query Execution events are used to identify slow queries that are related to database performance that can directly impact the Web server's performance.

Slow queries, if not pinpointed, can bring the server down by:

- Causing excess web server processes (Apache).
- Hang up queries in the database causing slower responses in the database.

These events do not require any additional configurations to the database. They are used in production to pinpoint performance bottlenecks in the database.

Description:

Generates an event whenever database related function execution raises above the given threshold. This event is directly associated to the "Slow Function Execution" event and is activated and defined in correlation with this event type.

Database functions that should be reported are defined in the "Slow Function Execution" function list (in additional rules).

Note:

To view supported databases, see the database related function prefixes listed in:
<install_dir>/lib/db_functions.txt or in windows <install_dir>\lib\db_functions.txt.

Inconsistent Output Size

Inconsistent Output Size events are used to verify that the page is rendering the same output to the client each time. If a page is not rendered the same each time, this means that some clients are seeing different output than others and some error has occurred.

This event is used mainly in production to indicate possible usability issues.

Description:

Generates an event whenever the output size is below or above the average output that is normally produced. The default values for this event type are set to 0. To generate events, configure these settings to a value that suits acceptable variance in percents from output to output of scripts.

Relative Events:

Event definitions are based on relative values i.e. percentage. Relative values are set according to warm-up settings, default value of 500 requests. If necessary, the default value can be modified manually by changing the zend_monitor.warmup_requests directive in the zend.ini.

Load Average

Load Average events are used to monitor the overall health of processes running on the server.

This event is used in production to highlight critical situations that might require analysis during high traffic situations.

Description:

Generates an event when the number of active processes waiting for CPU time, is higher than the number defined in the rule.

Default definitions are set to 0% for moderate and 0% for severe events. To start generating events these values need to be given a logical value based on the server's capabilities.

Custom Events

Custom events are a unique type of event that is provided for Zend Platform users in order to initiate events from their scripts.

This type of event is different than other event types in that it allows controlling event generation as opposed to the other events that are triggered by a certain occurrence.

Custom events are used to generate an event whenever the API function *monitor_custom_event()* is called from the PHP script.

Description:

This event type enables the generation of an event on occurrences that are not necessarily built-in Zend Platform events (error and performance issues). Custom events are used whenever you decide that it is significant to generate an event in a certain situation. Each event type is given a name for easy identification (\$type).

Function Usage:

```
monitor_custom_event(string $class, string $text[, integer $severe, mixed $user_data])
```

Parameters:

- **\$class** – helps to define several types of custom events. This description will be showed in the Event List and in the Event Details.
- **\$text** - error text used to describe the reason for the event. This text will appear in the Event Details.
- **\$severe** - the severity level of the triggered event, default value is Severe.
- **\$user_data** - adds a PHP variable that will be viewed in the Event Details (in Event Context-> Variables->User Defined). This forms the stored event data (similar to the information obtained in a PHP error event).

Aggregation takes place for these events when, two events occur in the same place and have the same \$class \$text \$sever(ity)

Note:

Action Rules defined for these events should be set to “send to URL” rather than “sending by e-mail” as there is only one definition for these events and event reports sent to a URL can be easily forwarded elsewhere. This is to prevent the overloading of e-mail. If we use the e-mail action, for every custom event, e-mail will be sent, and there can be many classes of custom events. However if the URL action is used, a script can be used to identify the event's class and different behaviors can be implemented according to class.

Define Event Actions

Once Event Triggers are configured the next logical step is to determine the actions and action rules that will be applied to Events generated by the PHP Intelligence module.

All Events are immediately reported inside Zend Platform's PHP Intelligence module. Events can be viewed from: PHP Intelligence | Event List. However, Events and the information included in the Event Details screen can also be sent via E-mail or to a URL by configuring Event Triggers.

Actions are applied to generated Events.

All Events are immediately reported inside Zend Platform's PHP Intelligence module.

Events can be viewed from: PHP Intelligence | Event List.

Event Actions enable sending Event details via E-mail or to a URL.

To configure Event Actions go to: Zend Central | Console and select Event Actions.

There are two steps to defining event actions. The first is to define "Actions and the second is to define "Action Rules".

- Actions determine the details of by specifying an e-mail address or a URL.
- Actions Rules determine which events by specific criteria will be sent according to one of the Actions.

Define Actions

Clicking the Event Actions URL opens the Actions dialog. This dialog allows you to define or remove Actions for the entire cluster.

Send data to	Remove
Email to amit.g@zend.com with subject Zend Platform Alert	
Send XML data to http://gollum/test.php	
Send XML data to http://gollum zend.office/var_dump.php	
Send XML data to http://gollum zend.office/ZendCanaveral/zps/zend_sen	

Figure: 10 - Event Actions

From the Actions screen you can:

- Add/remove global Action Types from a central administrative station
- View Action Types currently defined in the system

To add an Action:

1. Select one of the options from the Action Type drop-down list.
2. Depending on the selection e-mail or URL the action type details will change.
3. Enter the information according to the selected Action Type:
 - Target URL for the action type "Submit a report to the specified URL"
 - Recipient Address and Subject for the Action Type, "Send a report via e-mail."
4. Press Add to add the new Action Type to the "Current Action Types" list.

Note:

These Action Types can now be associated with Action Rules (see below). You can also change or remove the Action Type settings at any time.

Zend Platform supports two types of reports:

- **E-mail Report** - sends a text report to an e-mail recipient. This type of report is typically preferred by users who need to be notified of an event, but do not require the content of the report to be available for further use.
- **XML Report** - a structured XML report which is not only informative, but which can be made available for further use. For example, the .xml event data could be used as input for a monitoring script. The structure of the .xml report follows the structure shown below:

```
#each attribute exists if it exists in the Event Details screen
<?xml version="1.0" ?>
<event type event_id timestamp time severity>
#if there is an error:
    <error type>error text</error>
    <stats triggered_value avg load_average/>
#if there is a source file:
    <source file line/>
    <script name host uri>
        <vardata type name value/>
    </script>
#if there is a function:
    <function name>
        <args>
            <arg num value/>
        </args>
    </function>
```

```
#if there are included files:
<included_files>
    <file name\>
</included_files>

#if there is a backtrace for this event:
<backtrace>
    <call depth function file line/>
</backtrace>

</event>
```

Define Action Rules

The Define Action Rules screen is accessed from: Zend Central | Console.

This screen ties together the elements of the rule-based notification system (monitoring and reporting) by creating a logical rule that can be understood as follows:

When an Event of a user-defined Severity occurs in the user-designated Server, a specific Event Action (notification) will be invoked.

Active	Severity	Event Type	Server	Action	Remove
<input checked="" type="checkbox"/>	All	All	Gollum	Send XML data to http://gollum/test.php.	
<input type="checkbox"/>	Moderate	Database Error	fireforge	Email to amit.g@zend.com with subject: Zend Platform Alert	
<input type="checkbox"/>	All	PHP Error	All	Email to amit.g@zend.com with subject: Zend Platform Alert	

Figure: 11 - Define Action Rules

From this screen you can:

- Add/remove an Action Rule currently defined in the system.
- View Action Rules currently defined for an Action Type in the system.
- Edit an existing Action Rule and apply the changes.
- Disable an Action Rule.

To define Action Rules for a server:

1. Select the Define Action Rules tab
2. Enter Action Rule parameters in the Add a New Action Rule area.
 - a. Select an Event from the Events combo. (For a complete list of Events supported in the current version of Zend Platform, refer to the Configure Alert Rules section above.)
 - b. Select the severity from the drop-down list.
 - c. Select a Server from the drop-down list of servers currently defined in the system.
 - d. Select an Action from the drop-down list of Actions currently defined in the system.
3. Click Add or Apply.
 - a. Pressing Add adds the new Action Rule to the list of Action Rules defined in the system.
 - b. Pressing Apply applies the changes to a rule that you have edited.

Note:

Read more about how your organization can leverage information generated by events in the Tutorial - "Integrating Existing and Legacy Applications".

Configuring Performance

Customizing performance is a way to benefit from the Zend Platform advance performance features. Setting initial defaults enable the use of basic performance features. Additional configurations can be applied, to customize performance to correspond with organization-specific requirements. These configurations are addressed in the Chapter "Performance Lifecycle".

Performance Tab

Zend Platform's Performance settings are configured and viewed from: Performance |Console.

The Console section of the Performance tab is a main performance management screen through which basic details and commonly used Performance actions can be viewed as follows:

Code Acceleration	On	Reset	326 files accelerated 27.96 of 32MB used	Settings
Dynamic Content Caching	On	Reset	Default Cache Lifetime: 360 Default Dynamic Caching Conditions: GET vvv 1 file cached Add/Remove	Settings
File Compression	Off		No files are being compressed	Settings
Download Server	On			Settings

Figure: 12 - Performance Tab - Console

Initially the Console shows the installation defaults' regarding which feature is enabled (On or Off). Once changes are applied the console will be automatically updated with the new configuration settings (In some most changes are applied by restarting the Web-Server).

The following table lists the details and options available from the Console tab:

Component	Console Details	Actions
Overall Performance Gain	Shows the last performance test results.	Update - leads to Performance Testing Analyze Site. From this Tab site analysis tests can be run and results can be viewed. Get Latest Detailed Performance Gain - leads to Performance Testing Analyze Site, with the last performance test results expanded on the screen.
Code Acceleration	Shows the Code Acceleration component's status (On/Off) and basic code acceleration statistics.	Reset – Clears the Code Accelerator memory. Settings - Leads to the Code Acceleration section of the Settings Tab.
Dynamic Content Caching	Shows the Content Caching component's status (On/Off) and basic Content Caching statistics.	Reset – Clears the content cache. Settings - Leads to the Dynamic Content Caching section of the Settings Tab. Add/Remove - leads to Performance File View, where Cache settings can be added/Removed.
File Compression	Shows the Compression component's status (On/Off) and file compression settings.	Settings – Leads to the File Compression section of the Settings Tab.
Download Server	Shows the Download Server's status (On/Off).	Settings – leads to the Download Server section of the Settings Tab.

At the bottom of the Console there are shortcuts to individual Test functions as follows:

- **Run Performance Test** - runs a test that evaluates improved performance via Code Acceleration and Dynamic Content Caching.
- **Run Compression Test** - runs a test that evaluates improved performance via File Compression.
- **Run Download Test** - runs a test on a selected file that uses the Download Server (ZDS) to check the positive affect the ZDS has on performance.

Note:

These options lead to the appropriate option in the Testing Tab (Performance | Testing) and will not run the test before setting the preferences.

File View

Most performance configurations are done in the File View screen. Before describing the configuration tasks, it is important to understand the screen's layout and functionality.

The File View screen consists of two sections:

1. The Tree View on the left displays the list of directories and provides options for filtering the view (by status: Cached, Accelerated, Blacklisted and Compressed) the Virtual Hosts list is also updated from here.
2. The File View on the right lists files and their status, and also includes the different caching, acceleration and compression options that can be applied to selected files or to entire directories.

File Name	Lifetime	Conditions
testjava.php		
test.php	360	GET
sc_test.php		
rcv.php		
phpinfo.php		

Figure: 13 - File View - Tree View and File View Sections

Note:

Status changes made in the File View screen are immediately reflected on the screen. However, the actual changes take affect only after manually restarting the HTTP Server. A reminder to restart Apache, will appear on screen after changes are made and disappears only after restarting the server.

Clone Settings: To configure multiple servers or a server group with the same file settings: (1) configure a selected server and (2) then use the Quick Clone button to propagate settings from that server to other server nodes.

Tree View

The Tree View on the left displays the directories available under a selected document root. All the directories are listed by default.

The list includes filtering options to display directories by file type. To filter the list, select the file type from the drop down list.

The filtering options are: All Files, Cached Files, Acceleration Blacklist Files, and Compression Blacklist Files.

To refresh the list of files displayed on the right: click a directory's name.

Tree View - Virtual Hosts List

The Tree View lists all the directories and files in the default Document Root as well as any Document Root listed in the Vhost List. Displaying all the directories and files enables to view files included in the Document Root directly from Zend Platform and select files for Dynamic Content Caching.

Upon initial setup it is important to verify that all the applicable Virtual Hosts are included in the Virtual Hosts List for two distinct purposes:

1. To Benchmark test cached files. The Zend Benchmark (Performance | Testing) tests URLs per Virtual Host.
2. To update the File View option to reflect all Virtual Host's Document Roots.

The Tree View option maps all the Server's Document Roots providing a single view for displaying all the available directories and their contained files. Adding and deleting a Virtual host should reflect the actual Document Root activity on the server. (I.e. if you add/remove a document root from the server, you should add/remove its respective Virtual Host from the list).

Note:

The initial installation process creates a default Virtual Host list however; this may not include all the required virtual hosts and some may need to be added/removed.

Updating the Virtual Hosts List:

In order for Platform Performance to display files residing in a particular Document Root, you must add the Virtual Host to the list

The Virtual Host list in the File View reflects the current Virtual Host list as defined in Manage Cluster. You can update the list directly from the File View.

To Add or Remove a Document Root:

Go to Performance | Settings, and select Update Virtual Hosts to open the Virtual Hosts list.

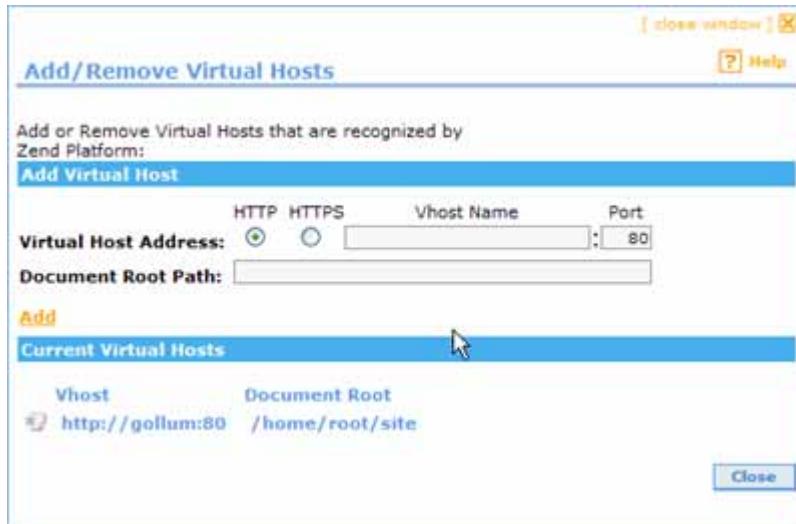


Figure: 14 - Update Virtual Hosts List

This screen includes two sections:

- Update Virtual Hosts List – add a Virtual Host
- Current Settings – Remove a Virtual Host and view current virtual hosts on the server.

Make sure that all the necessary Virtual Hosts are displayed in this list if not use "Update Virtual Host" to modify the list as necessary.

To update the Virtual Host list:

1. Go to Performance | File View and select Update Virtual Hosts List from the options at the top of the screen. This will open the Update Virtual Hosts List screen.
2. Specify the virtual host's details and provide an alias for the Virtual Host under Vhost Name.
3. Press Add to save the new Virtual Host and add it to the Virtual Hosts list.

File View

The File View displays files in a table, which can be sorted by column. The sorting options are: Status, File Name, Lifetime and Conditions.

File View - Dynamic Content Caching

Once all the Virtual Hosts have been established, and the default Caching Conditions have been set. Specific Content Caching settings can be applied to selected files or directories.

Content Caching activities include the following in chronological order:

1. [Define default caching settings](#)
2. [Modify file settings](#)
3. [Fine tune caching conditions](#)
4. [Define files to blacklist](#)

Define Caching Settings:

The File View screen lists all the directories and files in the default Document Root as well as any Document Roots listed in the "Vhost List".

Any cached file that has not been explicitly defined, automatically inherits the default cache settings

To open the File View screen, go to: Performance | File View.

The screenshot shows the Zend Platform File View interface. On the left, there's a sidebar with a tree view of Vhosts and Document Roots. The main area shows a table of files with columns for File Name, Lifetime, and Conditions. The table includes rows for testjava.php, test.php (cached, lifetime 360, GET), sc_test.php, rcv.php, and phpinfo.php.

File Name	Lifetime	Conditions
testjava.php		
test.php	360	GET
sc_test.php		
rcv.php		
phpinfo.php		

Figure: 15 - File View

The File view screen provides the following Full Page Content Caching options:

- **Cache** - Select files to be cached and press to enable content caching for the selected files.
- **Do not Cache** - Disables Content Caching for the selected files.
- **Define Cache** - Displays the "Define Caching Conditions" screen to add detailed configurations for selected files.
- **Update Blacklists** - Opens and modifies the current Blacklist.

-  **Clean** - Cleans cached file copies.
-  **Undo** - Cancels the last change to the settings.

Modify File Settings

To Modify File Settings:

1. Select a directory in the Tree View. The list of files residing in that directory is displayed in the File View.
2. Check the box next to the file(s) you wish to modify or the directory to select all files (and sub-directories) under it.
3. Click on the relevant icon in the toolbar.

Fine Tune Caching Conditions

Caching conditions can be changed per file or group of files.

To modify Caching Conditions:

1. Check the box next to the directory or cached file(s).
2. Click Define Cache to open the Define Caching Conditions Dialog. (Alternately click on a Cached file i.e. a file with the Cached indicator  next to it).
3. Apply Caching settings and press Save to save and close the dialog.

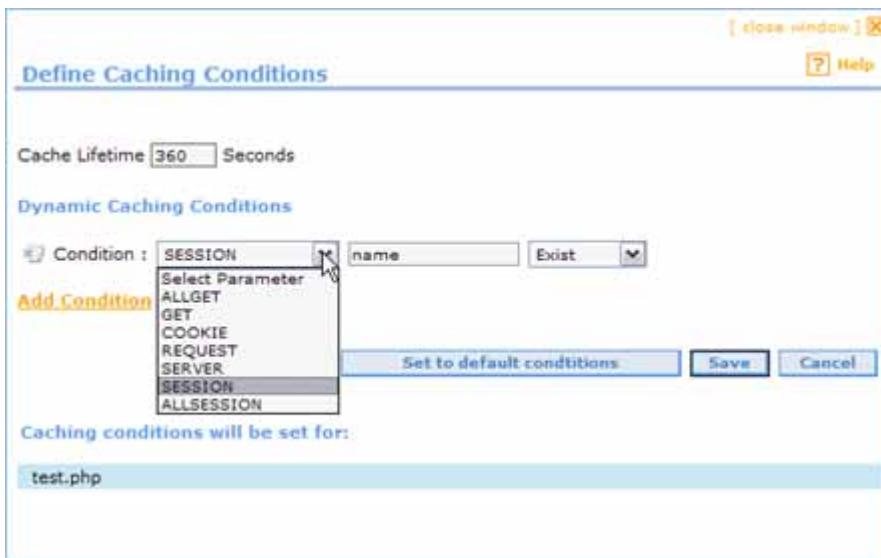


Figure: 16 - Define Caching Conditions

Modified settings will be displayed in the File View Tab next to the selected file/s. The Web server should be restarted in order to activate the new caching conditions. The restart server message will remain on the screen until the server is restarted.

The Define Caching Conditions dialog includes three buttons:

- **Restore Defaults** - Returns to the default caching settings.
- **Save** - The new settings are saved and are reflected on the screen but the changes will take effect only after restarting the server.
- **Cancel** - Cancels the new changes and returns to the previous settings.

Caching conditions may also apply to Variables stored in an Array.

Note:

Go to “Default Dynamic Caching Condition Parameters ” for a complete list of applicable conditions.

Note:

The zend_cache.ini file contains the list of all the files and directories assigned for Dynamic Content Caching including all the Conditions, as follows:

Use the File View to define the files and directories to be cached.

Do not edit this file manually!

zend_cache.lifetime=360

zend_cache.depends=ALLGET

zend_cache.path="/usr/local/apache/htdocs/hello.php"

zend_cache.lifetime=360

zend_cache.depends=COOKIE:my_cookie

A large cache.ini file can possibly result in slow performance. Therefore, it is recommended to un-cache (in the File View) any file deleted from the server.

Define files to Blacklist

The Blacklist separates acceleration and compression settings for files. With the blacklist users can prevent files from being accelerated or compressed.

Dynamic Content Caching

The concept behind Dynamic Content Caching is to store results of a first execution of a dynamically generated Web page. In this way, further requests made to the same page, will go to the Cache. Consequently, avoiding the overhead incurred by executing an application that renders output that does not change.

Zend Platform offers two ways to Content Cache files:

- **Full Page Content Caching** - For cases where it is possible to cache an entire output.
- **Partial Page Content Caching** - for cases where it is impractical or impossible to cache the entire output.

Note:

A separate tutorial has been included at the end of this guide to present Partial Page Content Caching functions and concepts (Please refer to the Tutorial: Partial and Preemptive Page Caching).

There are two caching conditions that can be applied to files:

- Default Full Page Content Caching settings can be applied to all files marked as cached in: Performance | Settings and going to the Dynamic Content Caching section of the settings screen.
- Specific Full Page Content Caching configurations can be applied to specific files by going to: Performance | File View.

Full Page Content Caching

Default Full Page Content Caching settings are applied to all files marked as cached in: Performance | Settings and go to the Dynamic Content Caching section of the Settings screen.

The content caching options are as follows:

- Dynamic Content Settings
- [Default Caching Conditions](#)
- [Default Dynamic Caching Condition Parameters](#)

Dynamic Content Caching	Current Settings	New Settings
Dynamic Caching Enabled	On	<input checked="" type="radio"/> On <input type="radio"/> off
Maximum Cache Size	671 MB	<input type="text"/> 671 MB
Minimum Free Diskspace	2133 MB	<input type="text"/> 2133 MB
Maximum Cached File Size	500 KB	<input type="text"/> 500 KB
Default Cache Lifetime	360 Seconds	<input type="text"/> 360 Seconds
Default Dynamic Caching Conditions	GET ???	Change Default Conditions

Figure: 17 - Dynamic Content Caching Settings

Dynamic Content Settings

The lifetime and conditions settings in the Settings tab are default values. These settings can be modified per file or per directory in the File View workspace.

Dynamic Content Caching Settings are as follows:

- **Dynamic Caching Enabled** - On The Dynamic Content Caching is active and working. Off – The Dynamic Content Caching is not in use.
- **Maximum Cache Size** -The maximum size allocated for cache. Occasionally and for short periods of time, this value may be exceeded but only until the next time that the Cache Cleaner deletes the files that expired. Set to "0" for an unlimited cache size.
- **Minimum Free Disk space** - The minimal reserved free disk space required. Reaching this value will end any further caching. The caching will resume as soon as the space is greater than this value.
- **Maximum Cached File Size** - The maximum allowed output cache file size. An output cache file that exceeds this value will not be cached. Set to "0" for an unlimited cache size.
- **Default Cache Lifetime** - The lifetime, in seconds, of cached data. The data will be re-generated if the cached version is older than the expiration time.

Note:

The Cache Cleaner is directly related to the directive `zend_accelerator.cache_cleaner_freq` that can be defined in the Configure PHP Settings screen. This directive defines when expired cache files are removed from the cache.

Default Caching Conditions

By default, Dynamic Content Caching, caches each request based on its full URL (ALLGET). You can condition the settings to be more general or more specific, as desired.

To change default caching conditions:

Go to Performance | Settings and go to the Dynamic Content Caching section of the settings screen. Select, Change Default Conditions to open the Define Default Caching Conditions dialog.



Figure: 18 - Define Caching Conditions Dialog

The default caching condition is ALLGET, which means that the entire GET string is used to identify a cached item. The GET string includes everything that appears after the question mark in a URL. (The ALLGET variables can be found in the `$_GET` PHP array as well).

The following actions and conditions can be applied to the Default Caching settings:

- To limit the ALLGET condition, select "Except" from the restrictions drop down list, to exclude a specific GET variable from the ALLGET.
- To change the ALLGET condition, select a new condition from the drop down list.
- To add another condition, press Add Condition and select another condition type from the list. Type the variables in the new condition row and restrict if necessary. The same condition can be used several times, each time with a different restriction.
- To remove any condition, click the delete icon next to the condition you wish to cancel.
- To change the Cache Lifetime's duration, type the new number (in seconds).

When all configurations are completed, press Save to save and close the dialog. Modified settings will be immediately displayed in the Settings tab. Click "Apply Changes" and restart the Web server to activate the new caching conditions. The message will remain on the screen until the server is actually restarted.

Note:

Caching conditions may also apply to Variables stored in an Array.

Default Dynamic Caching Condition Parameters

The following lists and describes each of the applicable parameters.

- **GET** - Indicates that you have selected certain GET variables. For example, consider the URL: `http://www.mysite.com/myscript.php?color=blue&size=L`. When set to ALL GET, a new request for `myscript.php?color=blue&size=M`, will not be taken from the cache and will be regenerated. If, however, the setting is changed to GET, with the value 'color', then the two URL requests would both be taken from the same cache content, regardless of the order of the variables in the request string. (The GET variables can be found in the `$_GET` PHP array as well).
- **COOKIE** - The Cookie variable is the variable given in the HTTP cookie. (It can be found in PHP `$_COOKIE` array as well). By selecting a cookie variable, it will also be considered a determining factor for cache hits, in the same way that GET variables are considered.
- **REQUEST** - The variable is set by the GET or COOKIE methods. (Can be found in PHP `$_REQUEST` array as well).
- **SERVER** - Server variable is set as a server environment variable. When selecting a server variable, (those listed in PHP `$_SERVER` array) it will also be used as a determining factor for cache hits, in the same way that GET variables are considered. To define a Server variable, select a variable from the list or choose Add a new variable to type in another variable.
- **SESSION** - The SESSION variable is useful when PHP sessions are in use. (Can be found in PHP `$_SESSION` array as well). Note: 1. If a script is cached using a SESSION variable and the session does not start in this script, the script will not be cached. 2. If a script is cached using a SESSION variable, yet the cookies are disabled on the user side and the SESSION ID is embedded directly into the URL, the caching will not take place.
- **ALLSESSION** - The script depends on all of the variables present in the session. (Can be found in `$_SESSION` PHP array as well).

Note:

It is mandatory to choose at least one Dependency.

File Compression

In order to maintain that Cached files improve overall performance, compression settings should be defined. These settings determine which files should be compressed. The mode of compression is gzip format – if the browser supports this format (If not, the data will be transferred un-zipped).

To Define Compression Settings:

Go to: Performance | Settings and go to the File Compression section of the settings screen. Select the file compression option that reflects your requirements.

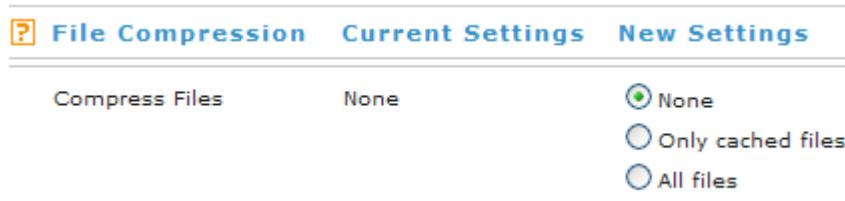


Figure: 19 - File Compression Settings

File Compression Options are as follows:

- **None** - File outputs are sent to the browser as is.
- **Only Cached Files** - Only the cached files are transferred to the browser in a gzip format—if the browser supports the format. If not, the data will be transferred un-zipped.
- **All Files** - Both accelerated and cached files are transferred to the browser in a gzip format—if the browser supports the format. If not, the data will be transferred un-zipped.

Recommended:

The recommended compression option is “Only Cached Files”, since the compression capabilities make use of the Dynamic Content Caching and there is no extra overhead for generating the compressed file (except for the very first time the URL is accessed.) Compressing accelerated files may cause some overhead and affect the overall performance. Use “All Files” if your main concern is improving the download time for the user.

Note:

There are some instances where it is preferable to deactivate compression for select files.

Compression can be deactivated in several ways:

- **Deactivate compression entirely** – should be done if the server is set to handle compression to prevent compressing files twice and rendering them unusable or when using PHP's compression feature zlib.
- **Setting compression to "cached files only"** – should be done when there is a large quantity of cached files and the rest of the files do not require compression.
- **Blacklist** – selectively disable compression for files do not require compression such as pictures that are already compressed or small files that do not require compression.
- Files under 1k are not compresses at all.

Blacklists

The Blacklist separates acceleration and compression settings for files. With the blacklist users can prevent files from being accelerated or compressed. The blacklist is accessed from: Performance | File View and pressing the Update Blacklist Files button.

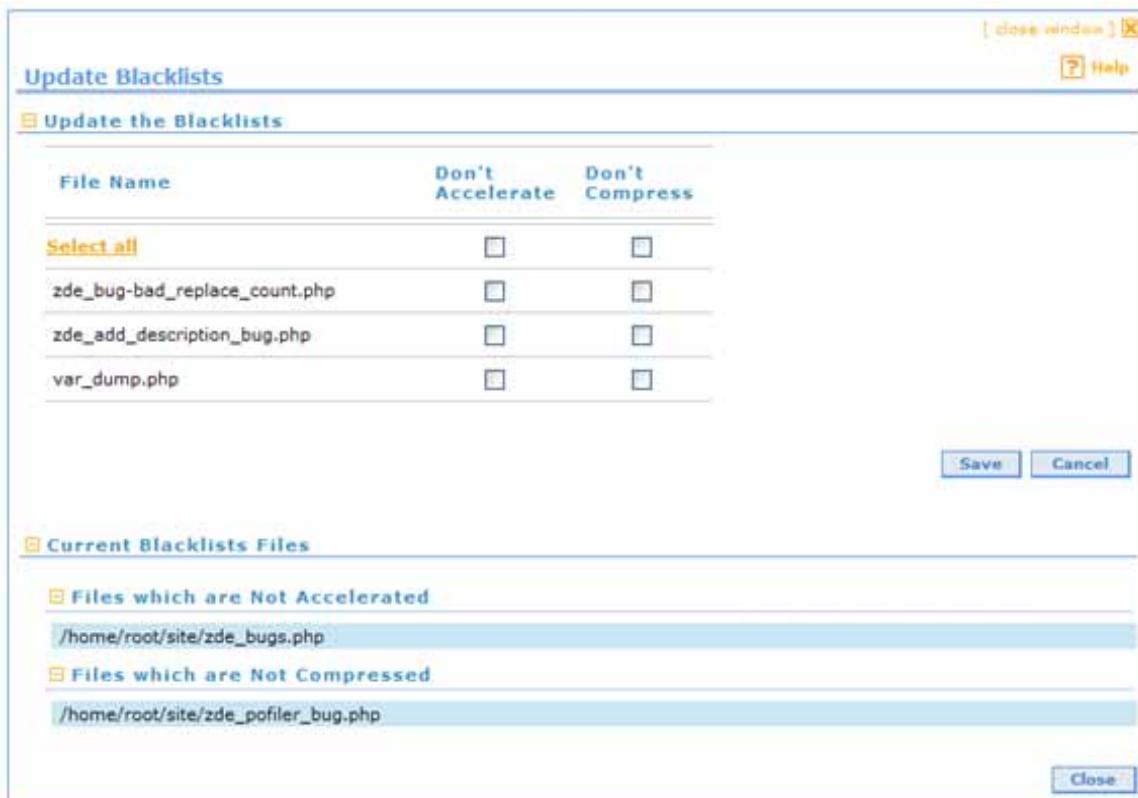


Figure: 20 - Update Blacklists Dialog

This dialog has two distinct sections:

1. **Update the Blacklist** – for defining blacklist criteria for selected files
2. **Current Blacklisted Files** – for viewing current blacklist settings

To update the Blacklist:

1. Select a file or files from the File View by selecting the check box next to the file names.
2. Press the Update Blacklist button.
This will open the Update Blacklists dialog with a list containing the selected files in the dialog.
3. The following options can be applied to each file:
 - Add a file into the Acceleration Blacklist - Check the 'Don't Accelerate' box.
 - Add a file into the Compression Blacklist - Check the 'Don't Compress' box.
 - Remove a file from a blacklist - Un-check the appropriate box.

- Add all the files to a blacklist - Check the appropriate box in the 'Select all' line at the top of the files list.

Current Blacklisted Files:

This section displays a list of files that are either not accelerated or not compressed or both. Files in the Compression Blacklist are not compressed (whether they are cached or accelerated).

To see the files in a blacklist, click on the Expand button.

Note:

Only single files (not directories) are added to the Blacklist.

Configuring the Zend Download Server (ZDS)

(This feature is currently not applicable for Windows Operating Systems)

The ZDS (Zend Download Server) is a PHP (Zend Engine) plug-in. The purpose of this plug-in is to efficiently deal with serving large, downloads. This is done to preserve website performance levels when handling large downloads that are served over the HTTP Protocol and consume bandwidth.

Downloads include, Video Files, Binary Products (such as .exe and .msi files), and other large files which are served over the HTTP protocol, and can potentially limit the performance of your website.

The ZDS provides two options:

1. Configure ZDS Settings
2. Test ZDS

ZDS functions in two modes:

- **Manual Mode** - Calling the API function `zend_send_file()` from PHP scripts.
- **Transparent Mode** – mapping file extensions to `zend_mime_types.ini`

Either mode can be run separately or in conjunction. Read on to find out how to configure the ZDS to run in either mode.

Manual Mode

In Manual mode, downloads are initiated by a PHP script that uses one all-purpose PHP function call. ZDS includes the PHP function `zend_send_file(filename)`. Calling `zend_send_file()` immediately starts the file download and terminates your PHP script's execution. This effectively frees up the Apache process to handle the next incoming request.

The `zend_send_file()` function can also serve files that are not under the Web server's document root. Furthermore, it can be used to run logical functions such as access restriction checks, before downloads are started.

Usability Example

If a download function is called `my_send_file($filename)`, you should integrate the `zend_send_file()` call in the following way in your source code:

```
if (function_exists("zend_send_file")) {  
    zend_send_file($filename);
```

```

} else {
    my_send_file($filename);
}

```

Alternate Method

`zend_send_file` can also be set to accept a second argument, the mime type of the file. This will override the default mime type setting.

The parameters are: `zend_send_file(string filename[, string mime_type])` and it would be called in the following way in your source code:

```

if (function_exists("zend_send_file")) {
    zend_send_file("/path/to/file.wma", "video/my-wma-type");
} else {
    my_send_file($filename);
}

```

Note:

If the `mime_type` is not specified or empty, the first mime type mechanism is used.

Manual Mode Usability Notes:

Do not create any output in Manual mode, before calling `zend_send_file()` - neither headers nor body - as this will interfere with the HTTP download.

Once you call `zend_send_file()`, the script terminates, so make sure all of your business logic runs before you call this function.

Sometimes files that are not under the same document root need to be served. Therefore, It is recommended to use the full path name to the file you want to serve. This will guarantee your script will work, even if you move it from your Web server's document root.

Transparent Mode

In Transparent mode, the file types that should be downloaded via ZDS are preconfigured, by mapping these files in the configuration file of your Web server. Files greater than the `min_file_size` directive will be automatically served by the ZDS.

To run ZDS in Transparent mode, make sure you meet the following requirements:

- The file extensions appear in the `zend_mime_types.ini` file and the file is mapped to the correct mime type.
For example: to serve .mpeg files via the ZDS, add the following line in `zend_mime_types.ini`:
 - video/mpeg mpeg
 - In your Apache Server's configuration file, map the file type to PHP.
 - For example, to map all .mpeg files to the ZDS in Apache by adding the following line to the Apache Server's configuration:
`AddType application/x-httpd-php .mpeg`

Both methods (manual mode and transparent mode) ensure that the Web application will continue to work even if, for some reason, you decide to temporarily disable the ZDS, (as

long as the ZDS module was loaded).

To Configure the ZDS:

Go to Performance | Settings and go to the Zend Download Server Settings section of the Settings screen.

P Download Server	Current Settings	New Settings
Minimum File Size	64 KB	<input type="text" value="64"/> KB
Apache Server MaxClients	Unknown	<input type="text" value="0"/>
Log File	/usr/local/Zend/logs/ZDS.log	<input type="text" value="/usr/local/Zend/logs"/>

Figure: 21 - Zend Download Server Settings

The settings screen provides three general ZDS configuration settings:

- **Minimum File Size** - The minimum size of files that will be served by the ZDS. Small files need not be served by the ZDS, since performance gain is insignificant. Default: 32Kbytes.
- **Server MaxClients** - The testing tool (in the GUI) uses this value to determine your server's MaxClients. Keep this value updated to the actual number of max clients of your server.
- **Log File** - The name and location of the log file where the ZDS reports completed downloads. Default: <install_dir>/logs. Make sure the directory exists and that the user who starts the Web server (usually root) has "write" permission.

Server MaxClients Recommendation:

The MaxClients setting depends on your server hardware. To achieve accurate test results the server should be set between 50-150 MaxClients. The MaxClients value must be the same in the Download Server Settings and the Web server's configuration file.

These settings are applied to downloads handled in one of the two handling modes: Manual and Transparent

Testing the ZDS

Once the ZDS has been configured a test can be run to check and analyze the overall efficiency.

- The default ZDS test uses the Manual mode of operation to invoke a PHP script, which sends a file of approximately 300KB.
- The same test tool can be used to check the Transparent mode. Make sure that you correctly map the file type you are checking in your Web server's configuration file - according to the configuration instructions.

The test simulates multiple requests for a specified URL, with and without ZDS. There are three sets of tests, each test is performed twice (once with the ZDS disabled and once with the ZDS enabled). These tests differ in the number of concurrent clients that simultaneously perform requests to the server.

Note:

It is extremely hard to artificially test ZDS. The main reason is that testing it on a LAN can easily saturate your local network, and if your MaxClients is very high, Apache Benchmark (ab) may have difficulty handling the concurrency. For this reason, it is recommended to test ZDS with a relatively low MaxClients setting (e.g., 50-150) so that you don't reach any of these limits. The ZDS includes a version of ab, which was modified to support bandwidth throttling, which is used by the testing tool.

Caution:

During a test, your Web server will be fully loaded. A test can take several minutes so you should run it on a development machine or on an offline production machine.

To Test the ZDS:

Go to Performance | Testing and go to the Test Download tab.

The screenshot shows the Zend Platform Performance - Testing - Test Download Tab. At the top, there are links for Email, Print, Help, and the date/time (05 Feb 2006, 17:03:12). It also shows the user is logged in as Admin and the server name is Gollum. Below this, there are three tabs: Test URL, Analyze Site, and Test Download (which is selected). The Test URL for testing is set to http://gollum:80/ZendPlatform_2_1_2/zps/zend_send_file.php. Under Clients' Bandwidth, the option for DSL (0.5 Mbit/s) is selected. The Number of Max Clients on the Server is set to 0. A note below states: "NOTE: Please make sure that the value for the Max Clients is the actual MaxClients value of the server (it can be found in your Apache httpd.conf file)." There is a Run button and a link to Show Last Download Test Report.

Figure: 22 - Performance - Testing – Test Download Tab

The Test Download Tab consists of two sections: The test options are on the upper section and the test results appear below (after running a test or displaying the last test results).

Running a Test

1. Type in the URL you want to test.
The default test is a PHP script which uses `zend_send_file()` to send a 300K zip file (Testing very large files will take a very long time).
2. Choose the bandwidth limit you want to simulate for the clients. For a faster test, select a higher bandwidth (You cannot choose full bandwidth because your network card will be saturated, making the test irrelevant. The test tries to simulate a typical Internet server that has clients connected either by ISDN or DSL).

3. Enter the number of maximum clients that your server can handle. Use the precise value by checking the value of the MaxClients directive in your server's configuration file (The ZDS tries to identify your MaxClients value in the installation process, via the httpd.conf file, which is the GUI's default value. However, this value can be changed after the installation; and should be double-checked. Using an inaccurate MaxClients value, may not present accurate results).
4. Click Run.

Understanding Test Results

Once the tests have completed, you will see two tables and graphs with results that show Requests per Second and Average Time per Request for each test run.

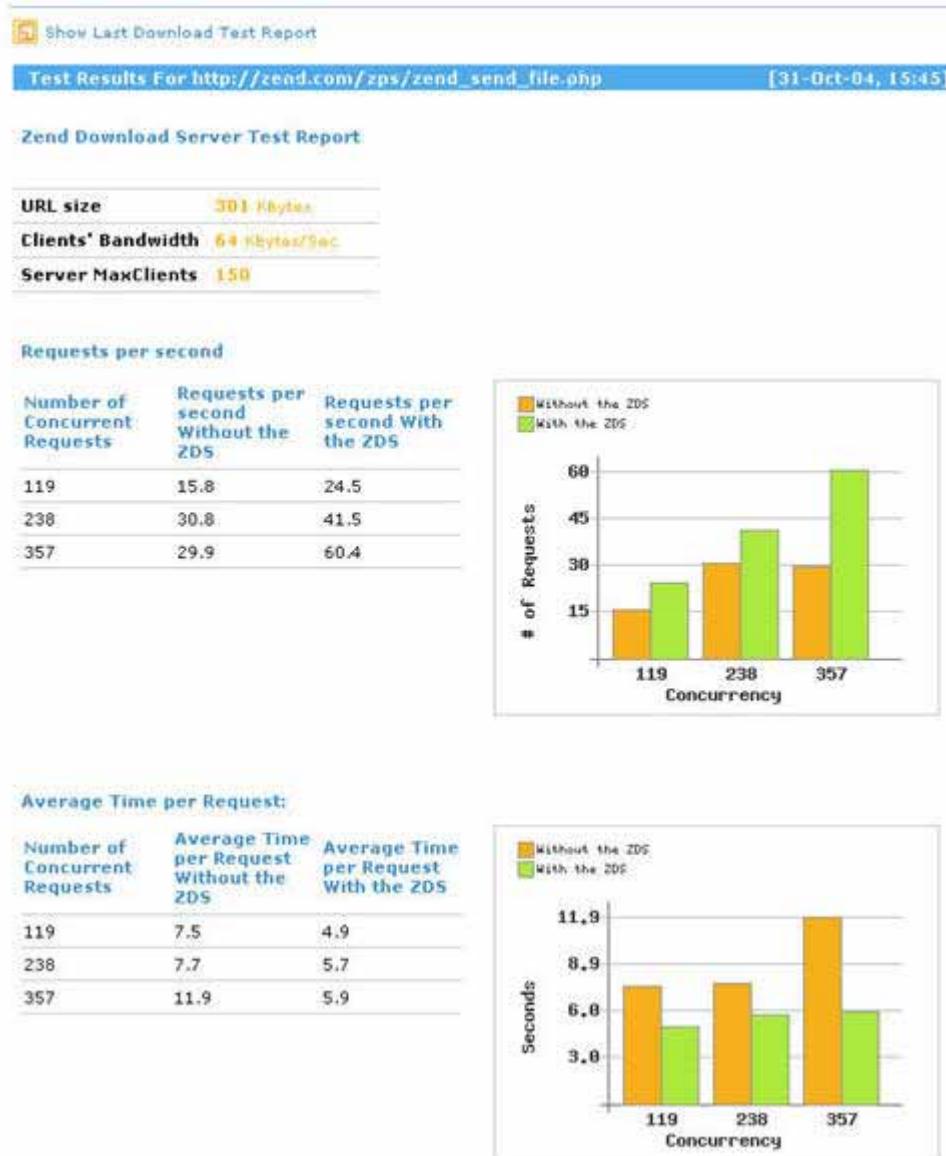


Figure: 23 - Zend Download Server - Test Results

When the Zend Accelerator is disabled, only cached scripts are tested.

Note:

If you do decide to run the ZDS Test on a production server, you can watch the log file to see how many concurrent jobs ZDS is handling. This indicates the number of Apache processes that would have been used if the ZDS were not installed.

Clone Wizard

Once all settings have been configured on a node, the Clone Settings feature allows for applying these settings to the different nodes in one single process.

To clone configurations:

1. Go to Zend Central | Console and click Clone Wizard.
The Clone wizard dialog opens after selecting a source server.
2. Use the **View Tree By field** to quickly find the source server.
3. Select the Source Server—the server whose configuration settings you wish to distribute to other servers in the network—from the list of servers in the tree display.
4. Click Next to open the Configuration Selection screen.

Note:

The Configuration Selection screen is a collapsible list of configuration parameters to be taken from the source server that was selected. Only selected parameters are cloned. By default, all configuration parameters are deselected.

5. Select the PHP configuration, Zend products settings or entire Configuration file (of the Zend products) for the source server with the settings you wish to clone.
6. Select additional configuration parameters you wish to clone.
7. Click Next, to select the destination server or servers.
8. In View Tree By field, select the tree view you wish to use to find the destination server quickly.
9. Select the Destination Server(s) – i.e. the server(s) with the configuration settings that you wish to change to match parameters selected from the source server (from the list of servers in the tree display).
10. Click Clone.

The Clone Wizard distributes the selected parameters from the source server to the destination server(s). A Broadcast Summary screen appears indicating that the configuration settings have been cloned to the selected destination servers.

User Management

Granting different levels of permissions to different users provides a means for controlling actions performed in the environment and enforcing work procedures. This is the last step to customizing Zend Platform to suit your working environment.

Zend Central's User Management tab includes Platform's multiple users functionality. This feature set allows different users to login to Platform.

Each user has a set of permissions that are defined by the system administrator that determine:

- Data the user is allowed to view
- Zend products the user is allowed to access
- Actions the user is allowed to perform

The User Management workspace displays information about the users currently defined in the system. It also provides shortcuts to the User Management functions supported by Zend Platform.

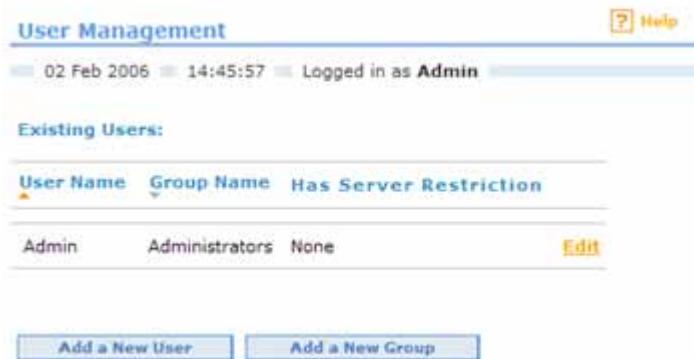


Figure: 24 - User Management

The information fields and functions that make up the User Information workspace are as follows:

- **Logged in as** - Displays the name of the User who is currently logged in to Zend Platform.
- **Current Existing Users** - A list of users currently defined in the system.
- **Group Name** - Users are defined within the system as belonging to a particular group—and not as independent entities. This is the name of the Permission Group to which the User belongs.
- **Handle Groups** - Selects the group whose attributes you wish to edit or remove entirely.
- **Edit/Remove** - Allows you to edit or remove entirely the settings for a specific User of a specific Group.
- **Add a New User** - A shortcut to the Add a New User Wizard.
- **Add a New Group** - A shortcut to the Add a New Group Wizard.

Press here to see how to use user permissions to restrict access to virtual hosts.

Adding and Editing Users

Zend Platform allows you to Add/Edit Zend Platform users. However, two conditions must be met:

1. The master user adding new users must have administrative level permissions in the system.
2. New users must be added to an existing group. Users are defined as belonging to a particular group - not as independent entities.

Adding a User

Zend Platform allows Administrator-level users to create new users.

To add a new user:

1. Click the Add a New User button in the lower left corner of the User Management workspace.
The Add a New User wizard opens.
2. In the Add New User wizard, define the following General User Settings:
 - **User Name** - Enter a User Name that the User will use when logging in to Platform.
 - **Password** - Enter a Password that the User will use when logging in to Platform. Platform GUI Passwords, may contain, between 4-16 characters including the following: the alphanumeric characters 'a' through 'z', 'A' through 'Z' and '0' through '9' and the special characters (-) dash, (_) underscore and (.) period.
 - **Confirm Password** - Confirm the Password.
 - **Permissions Group** - Select the Permissions Group to which the New User will be assigned from the list of Permissions Groups that are currently defined in the system.
3. Click Next to go to the second step.
4. In the Add New User wizard (2) Select the servers that you wish to allow the New User to access.
5. Click Finish.
The new user will be created in the database, with the defined permissions.

Note:

Added/Edited users can only view and edit events for the servers the User has permission to access. Conversely, servers denied to the User will not appear in the server tree when the User logs in to the system.

Editing a User

Zend Platform allows administrator-level users to edit the preferences for any User currently defined in the system. Non-administrator users can use this option to modify their password.

To edit user preferences:

1. Click the Edit button to the right of the user whose preferences you wish to edit.
The Edit User Wizard opens with the user's name appearing in the User Name field.
2. In Edit User wizard – Step 1, define the following General User Settings:
 - **Password** - Enter a Password only if you wish to change the Password for that User; otherwise leave the Password field empty.
 - **Confirm Password** - Confirm the Password only if you are changing the Password for the User.
 - **Permissions Group** - Select the Permissions Group to which the User will be assigned from the list of Permissions Groups that are currently defined in the system.
3. Click Next to go to the second step.
4. In the Edit User wizard (2) screen, select the servers that you wish to allow the User to access.
5. Click Finish.
The changes will be applied to the User settings and saved in the database.

Adding and Editing Groups

Add/Edit a Group

Zend Platform allows Administrator-level users to create new groups and define user access permissions.

To create a New Group:

1. Click the Add a New Group button in the lower left corner of the User Management workspace to open the Add a New Group dialog.
2. In the Add New Group dialog, select the preferences and permissions to assign to the New Group from the table.
3. Enter a name for the New Group.

Note:

Remember to click Save to save changes or create a Group in the database and register its settings.

The following is a list of group preferences and their definitions:

- **Group Name** - Enter a name for the New Group
- **Delete an event** - Enables the Delete Event option, for deleting events in the Event Details screen and the Event Window.
- **Ignore an event** - Enables the Ignore Event option, for ignoring events in the Event Details screen and the Event List Window.
- **Close an event** - Enables the Close Event option, for closing events in the Event Details screen and the Event List Window.
- **Reopen an event** - Enables the Reopen Event option, for reopening events in the Event Details screen.
- **See the event internal data in Event Details** - Allows users belonging to group to view the event internal data (variables, included files) from the Event Details screen.
- **See the event source code in Event Details** - Allows users to view the event source code in the Event Details's embedded viewer.
- **Use the Zend Studio Diagnostics in the Event Details** - Allows users to view, profile and debug event source code in Zend Studio.
- **Changing Zend Platform Settings (in the Preferences)** - Allows users to change preferences in Zend Central | Preferences.
- **Configure and change the Event Actions** - Allows users to configure and change Event Actions from the Event Actions screen.
- **Configure and change the Action Rules** - Allows users to configure and change Action Rules from the Define Action Types/Rules screen.
- **Manage Server** - Allows users to manage servers from the Manage Cluster screen.
- **Manage Groups** - Allows users to add and change group settings from the Manage Clusters screen.
- **Cloning configuration between servers** - Allows users to clone server configuration to additional servers from the Clone Wizard.
- **Update Servers Data** - Allows users to update server data from the Update Servers Data screen.

- [Use Support Tool](#) - Allows users belonging to the group to access the Support Tool from the Support Tool link.
- [Configure and change the Event Triggers](#) - Allows users to configure and change Event Triggers from the Event Triggers screen.
- [Configure and change the PHP settings](#) - Allows users to configure and change PHP settings from the Configure PHP Settings screen.
- [Go into the Performance section of nodes](#) - Allows users to access the Performance Tab for nodes.
- [Go in the Java Bridge section of nodes](#) - Allows users to access the Java Bridge Tab for nodes.

User Settings

User settings are retained in the system in several ways:

- User Group settings that are stored in the configuration database.
- Zend Central (User Management Tab) remembers each user's last settings. The user's last settings automatically populate the component fields, when opening any of the sub-screens and dialog boxes that make up the User Management Tab.

This completes the Configuration chapter of the User Guide. In this section we have described the different configuration tasks that can be done. In the next chapter we will describe how to implement Zend Platform in the working environment by using PHP Intelligence to create a Problem Resolution lifecycle.

Please refer to "Appendix B – Configuration Check List" to read/print the Zend Platform Configuration Check List that summarizes the configuration tasks.

Chapter 3 - PHP Intelligence

IN THIS CHAPTER...

THE PROBLEM RESOLUTION LIFECYCLE

CREATING EVENTS

FINDING EVENTS THAT INTEREST YOU

UNDERSTANDING EVENT DETAILS

CONTROLLING INFORMATION DISPLAYED IN AN EVENT

CUSTOMIZING EVENTS

DATABASE MAINTENANCE

In this chapter we will discuss how PHP Intelligence can provide improved communication between developers, managers and QA teams through implementing the Problem Resolution Lifecycle.

The Problem Resolution Lifecycle

Background

Developing and maintaining Web applications is an intricate and highly demanding process. Zend Platform facilitates the intricacies of the development process by employing an efficient problem resolution infrastructure – “the Problem Resolution Lifecycle.” This infrastructure’s main goal is to help make the most out of challenging environments and tight schedules and prevent problematic issues from falling between the cracks.

With the Problem Resolution Lifecycle, organizations can improve communication between the development, testing and IT teams to streamline the development and deployment processes.

Using PHP Intelligence in development and production environments unifies the working environment and ensures improved information collection and distribution between development teams, testing teams and IT teams (See illustration below).

Zend Platform in the Working Environment

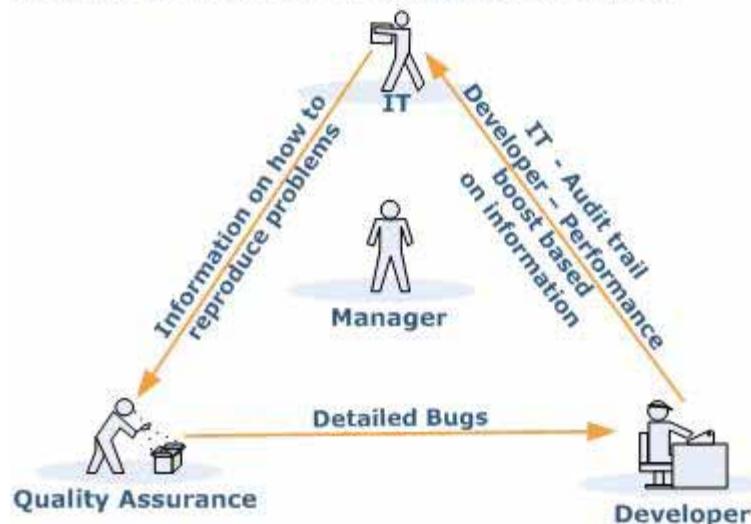


Figure: 25 - Problem Resolution Lifecycle

Using Zend Platform in your working environment ensures that pertinent and focused information reaches the right person at the right time. The enhanced information exchange results in major improvements in quality of code, time to production and overall performance and stability. The subsequent benefit is more resources dedicated to activities focused on improving and expanding the current application and less time spent on locating information necessary for recreating and resolving code and performance issues

In the Problem Resolution Lifecycle, PHP Intelligence assists the efforts of the development, testing and IT teams to quickly pinpoint, analyze, and resolve issues such as: PHP Slow Script Execution, Function Errors, Database Errors etc.

Zend Platform Problem Resolution Lifecycle



Figure: 26 - Problem Resolution Workflow

Zend Platform's PHP Intelligence functionality is enhanced by:

- Implementing customized Event Rules to areas prone to problems in your unique environment – facilitating focused and efficient problem resolution.
- Analyzing “Full Problem Context” grants a detailed insight of problematic occurrences.
- Integrating with Zend Studio to resolve problems with state-of-the-art development and debugging tools.

Implementing the Problem Resolution Lifecycle

The Problem Resolution Lifecycle is a process of defining PHP Intelligence Event Triggers according to acceptable run-time, performance parameters. PHP Intelligence enforces Event Triggers and issues Event information in an Event Details screen according to the Event Trigger definitions. When an Event occurs, PHP Intelligence compiles a complete profile of the Event's occurrence and its precise details. An Event Details screen includes comprehensive details in order to enable developers and testers to recreate the Event in a way that mirrors the conditions of the original occurrence. This information can then be used to diagnose problems by fine-tuning Event Triggers to accommodate normal occurrences or resolve actual run-time problems and errors.

With Zend Studio Diagnostics, problems and errors can be easily diagnosed using the Event Details screen functions, Test URL and Profile URL, and further information can be analyzed using Debug URL and Show Source Code. In addition, problems in code can be immediately resolved using the Zend Studio Editor which allows changes to be immediately made and deployed, not only to a single server but also to all nodes belonging to the same Group.

Events can be preserved to leave an indicator of these occurrences if necessary. Furthermore, user permissions can define who is permitted to perform actions inside an Event Details screen; enforcing a structure that encourages communication between the different teams.

Creating Events

Event generation is an out-of-the-box feature. Directly after installation, Zend Platform's PHP Intelligence will begin to monitor events according to Default Settings. To further enhance the effectiveness of PHP Intelligence, events thresholds can be customized. In a similar manner thresholds can be gradually modified to not only reflect improvements in performance but also to verify that problematic issues have been resolved.

Configuring Events

Events can be configured according to each environment's specific requirements. The main configuration changes that should be done are to do with tuning Event Trigger values and defining a list of Functions and PHP errors to be monitored.

To Configure Event Triggers, go to PHP Intelligence | Event Triggers and change the default settings according to your requirements.

A help button  appears next to each Event Type. Pressing this button will display a description of the selected Event and the Event's parameters. (Alternately go to Choosing and Defining Event Triggers)

Disabling Events (Triggers)

In some cases there may be Events that are either not applicable to your system or unnecessary. Events are disabled from the PHP Intelligence module. When an event is disabled the event will not be monitored and no event information will be stored.

Disable Event Triggers

To disable Event Triggers go to PHP Intelligence | Event Triggers and select "Configure Event Triggers."

In the Define Event Triggers Table, the Check box in the Active Column indicates if an Event Type is monitored or not.

To prevent a selected Event from being monitored, disable the Rule by unmarking the Check Box. This will deactivate and stop collecting event related information.

List Entry of Watched Functions

Zend Platform allows you to monitor a list of functions by referencing a text file that includes the functions you wish to monitor. Users who must monitor large numbers of functions will find this method of defining watched functions a convenient alternative to editing the php.ini file line by line.

Use the following PHP functions to reference a text file containing the list of functions to monitor.

The following function is typically used to create a list of functions to watch. It forms part of the php.ini file.

`zend_monitor.watch_functions=mysql_connect,mysql_query`

The following function refers zend_monitor.watch_functions to a text file at a specific location. This file contains the list of functions to monitor.

Unix:

```
'zend_monitor.watch_functions=@/usr/local/Zend/lib/watch_funcs.txt'  
'zend_monitor.watch_results=@/usr/local/Zend/lib/watch_res.txt'
```

Windows:

```
'zend_monitor.watch_functions="@C:\Progra Files\Zend\ZendPlatform-  
2.2.0Beta\lib\watch_funcs.txt"'  
'zend_monitor.watch_results="@C:\Program Files\Zend\ZendPlatform-  
2.2.0Beta\lib\watch_res.txt"'  
(In Windows the quotes must be present)
```

The text file should contain one function name per line.

Example:

```
mysql_connect  
mysql_pconnect  
mysql_query  
mysql_db_query  
mysql_unbuffered_query
```

User functions can also be included in the Watch Functions file. Each user function must be added with its Class (class::function).

If necessary, inheritances should also be included in the file as only functions explicitly specified in the Watched Functions file are watched.

Finding Events that Interest You

Zend Platform provides several ways for viewing Events that occur; each way has different advantages and can be used to suit different requirements as follows:

- [Zend Central | Console | Events at a Glance](#) displays the top five events that occurred, on all the servers. Double clicking on an Event in the Console opens its Event Details screen.
- [PHP Intelligence | System Health](#) displays an up-to-date “snapshot” of events monitored by Zend Platform and listed by server and Event Type. Selecting a Location (Node) or a specific Event Type automatically filters the view to display relevant Events in the Event List.

- [PHP Intelligence | Event List](#) is a filterable display for viewing events in a table according to various parameters. Choosing “Change Table Fields” modifies these parameters. This opens a selection list of all the possible field options. This window also includes an option to locate events by their Event ID.
- [Zend Central | Console | Configuration and Management Tools | Event Actions](#) for sending events of a certain type to e-mail or URL according to predefined rules. This provides a proactive means for sharing information either with parties that need to be informed when certain events occur (e-mail) or for integrating event information to other applications (URL).
For example: a manager may only want to know about Severe PHP errors that indicate some or all of the Web application is not working. Setting an Event Action to send event information by e-mail means that this manager is immediately informed of the event, as long as the e-mail account is accessible

Note:

Read more about how your organization can leverage information generated by events in the Tutorial – Integrating Existing and Legacy Applications.

System Health

The most comprehensive way to view and manage events is through the System Health screen. The System Health sub-screen provides an up-to-date "snapshot" of events monitored by Zend Platform. Events are displayed in a filterable table, which is updated by manually refreshing the browser (pressing Refresh) or by setting automatic refresh (Zend Central | Preferences). The System Health table includes events categorized by location (server/group) and event type.

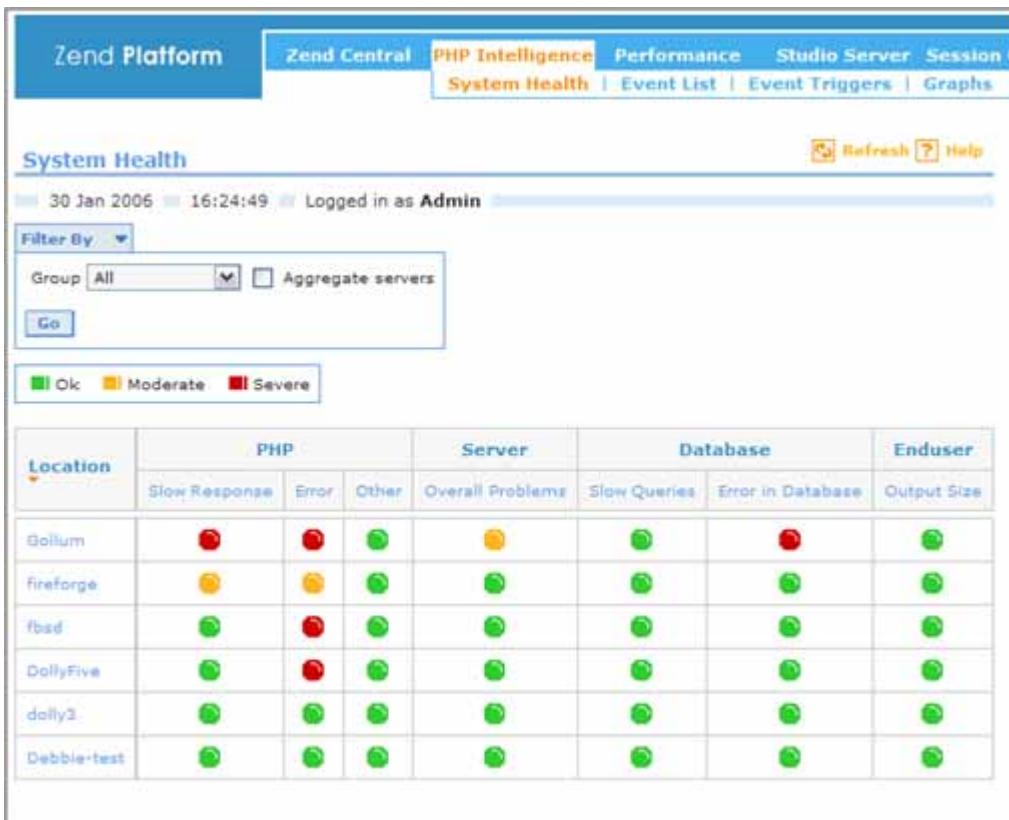


Figure: 27 - System Health Screen

To view the System Health table:

1. Go to PHP Intelligence | System Health.
2. Select the "Filter By" option to filter events displayed in the table.
Filter options are described in detail below.
3. Click Go to display the System Health table.

The fields that make up the Event Summary table are as follows:

- **Location** -where the event occurred.
- **PHP** -Current PHP events for the selected Host.
- **Server** -Overall events related to general performance of server.
- **Database** -Database events related to handling of queries.
- **End User** -The output site; field includes status information about output site.

Filtering Table Data

Zend Platform allows you to filter the event information displayed in the System Health table. There are three group options:

- **All** - displays System Health information for all servers (and groups).
- **Ungrouped** - displays System Health information for servers that do not belong to a specific group.
- **Server by name** - displays System Health information for the selected server group only.
- **Aggregate servers** - aggregates the servers belonging to the same group into one row.

To filter the data displayed in the System Health table:

1. Select the Filter By options—Group and/or Event Name—you wish to apply to the table.
2. Click Go to display the filtered events in the System Health table.

Event Reporting

Event reporting is the process of identifying events that were generated for the same reason and can be reported X times rather than reporting multiple occurrences of the same event.

There are two options for reporting events:

1. By Server un-aggregated)
2. By Group (aggregated)

Aggregation by server happens when identical events are generated on different servers. The same event is reported (for each server) separately. This is the default.

Aggregation Groups

Event aggregation / reporting is the process of identifying events that were generated for the same reason and can be reported as one event that occurred X times rather than reporting multiple occurrences of the same event.

There are two options for aggregating events:

1. By Server
2. By Group

Aggregation by **group** happens when several servers are placed in a group that is defined as "Aggregated" (Console | Manage Cluster | Manage Groups).

At the bottom of the System Health table there is a message stating that:

(*) Events that took place on the server before the server was a member of an aggregated group.

Aggregation by **server** happens when identical events are generated on different servers. The same event is reported (for each server) separately. This is the default.

Event information can only be aggregated at the time of the occurrence and for information integrity purposes will not be retroactively aggregated. If events occurred on individual servers before they were grouped, they will be reported as un-aggregated events.

Event List

The Event List is a filterable table that displays Events that occurred within a user definable period of time.

Information displayed in the Event List can be displayed as follows:

- Events can be filtered by the following categories: Events From, Event Type, Virtual Host, (Event) Severity, Status, and Time Filter.
- Events can be located in this screen by the event's ID.
- Columns in the Event List table can also be customized (Use Change Table Fields).

Specialized filters can be created and saved using the Manage Filters option.

Filtering by Virtual Host

Several servers can be added to a user-defined group and used to filter the display to show Events that occurred on a specific selection of servers.

To define a virtual Host name that will be added to the list in the filter, go to the Virtual Host filter field and choose "Selected". This option opens the Change Virtual Host screen.

The Manage Filters section allows users to save user defined filter definitions.

To save user defined filter definitions:

1. Define the filter settings in the Filter By section.
2. Select Manage Filters to expand the filter management options.
3. Enter a user defined name for the filter and press Save to add the new filter.
4. Use Load Filter each time you want to use a user defined filter.
5. Use Remove Filter to delete filters from the Load Filter List.

The filter section of this screen also includes a search field for searching for a specific Event by ID.

Zend Platform also allows you to change the table fields displayed on-screen. (Press The Change Table Fields button and select the fields you want to view from the list).

Working with the Event List

To view the Event List:

1. Go to PHP Intelligence | Event List.
2. Select the "Filter By" option you wish to apply to the table.

- Filter operations appear as follows:
 - **Events From** - Filter Events according to grouping definitions. These definitions list events according to where they were generated (in the PHP, Database) or what type of event (slow response, error). The values are: All, Bandwidth-Other, Web server-Other, PHP-Slow Response, PHP-Error, PHP-Other, Database-Slow Response, Database-Error
 - **Event Types** - Filter Events displayed in the table according to event type. The values are: All, Slow Script Execution (absolute), Slow Script Execution (relative), PHP Error, Function Error, Slow Function Execution, Excess Memory Usage, Database Error, CPU Load, Inconsistent Output Size, Custom Events.
 - **Virtual Hosts** - View Event information for either All hosts or selected hosts. Click Selected to open the Change Virtual Host Selection dialog box. All, Selected or predefined user selections.
 - **Severity** - Filters Event information according to severity. The values are: All, Moderate, Severe
 - **Status** - Filters information displayed in the Event List according to the Event's handling status. The values are: All, Opened, Closed, Ignored
 - **Time Filter** - Filters information displayed in the Event List according to a user-defined time frame. The values are: All, Past Hour, Past Day, Past Week, Past Month
 - **Find Event by ID** - Finds (and displays) an event with a specific ID number. (By entering an event id and pressing Find). The values are: Sequential numbers that were assigned to Event Details.

3. Click Go to display filtered events in the Event List table.

The screenshot shows the 'Event List' page of the Zend Platform i5/OS. At the top, there are navigation links: 'Change Table Fields', 'Refresh', 'Help', and 'Logout'. Below this is a toolbar with buttons for 'Find', 'Save', 'Remove', and 'Go'. The main area displays a table of event logs with the following columns: Id, Event Type, Count, Last Occurrence, Location, Vhost, URL, Source File, and Line. The table contains five rows of data. At the bottom of the table are buttons for 'Select All', 'Delete Selected', 'Ignore Selected', 'Close Selected', and 'Preserve Selected'.

	Id	Event Type	Count	Last Occurrence	Location	Vhost	URL	Source File	Line
<input type="checkbox"/>	64	PHP Error	2	18 Jan 2006 01:16:39	kuzya	kuzya	.../my_test.php	.../my_test.php	3
<input type="checkbox"/>	61	PHP Error	1	16 Jan 2006 17:01:31	dolly2 zend.office	dolly2 zend.office	.../event_creator	.../event_creator_w.php	364
<input type="checkbox"/>	62	Excess Memory Usage (Absolute)	1	16 Jan 2006 17:01:31	dolly2 zend.office	dolly2 zend.office	.../event_creator	.../event_creator_w.php	
<input type="checkbox"/>	52	Inconsistent Output Size	4	16 Jan 2006 17:01:30	dolly2 zend.office	dolly2 zend.office	.../size_script.ph	.../size_script.php	
<input type="checkbox"/>	53	Excess Memory Usage (Relative)	4	16 Jan 2006 17:01:30	dolly2 zend.office	dolly2 zend.office	.../size_script.ph	.../size_script.php	

Figure: 28 - Event List

The Following list describes the fields that make up the Event List.

- **ID** - Sequential number assigned to an event.
- **Event Type** - A descriptive name assigned to the event.
- **Count** - Number of occurrences of the event.
- **First Occurrence** - Date and time of event's original occurrence.
- **Last Occurrence** - Date and time of event's most recent occurrence.
- **Location** - The name of the server or Aggregated Group where the event occurred.
- **Vhost** - Name of the Vhost where the event occurred.
- **URL** - The URL where the event occurred.
- **Source File** - Path to PHP source file.
- **Line** - Line in code where event occurred.
- **Aggregation Hint** - The hint in the code that caused the event to be aggregated.
- **Function Name** - Name of PHP function where event occurred.

- **Status** - Status of the event.
- **Severity** - Severity of the event.

The following actions can be performed on the events in the Events Table:

To apply one of these actions to an event/events select the event by checking the check-box next to the events:

- **Delete Selected** – Deletes the Events form the database.
- **Ignore Selected** – A new Event Details screen will not be created for this specific occurrence Additional occurrences of this event will be added to the original Event's details.
- **Close Selected** – Changes the status of the event to closed and preserves the Event in the Database. If the same event occurs again a new Event Details screen will be created.
- **Preserve Selected** – Saves the Event in the Database until the next scheduled cleanup.
- **Reopen Selected** - Changes the status of ignored or closed events to open.

Note:

The default value for database cleanup is 7 days.

Depending on the operating system database cleanup settings can be changed in:
/usr/local/Zend/etc/php-embed.ini or <install_dir>\etc\php-embed.ini, by setting a different value for the parameter zend_monitor.event_lifetime

Event Details

Understanding Event Details

Event Details are generated in accordance to Event Triggers. Event Details are a diagnostic tool that provides a complete audit trail and options for investigating and resolving events.

Event Details are viewed in several ways. The regular way of viewing events is from Zend Platform. Events can be configured to be sent by e-mail recipients or to a URL (in XML format). However, Event Details always include the same information regardless of the viewed output (Regular Event Details, XML or e-mail).

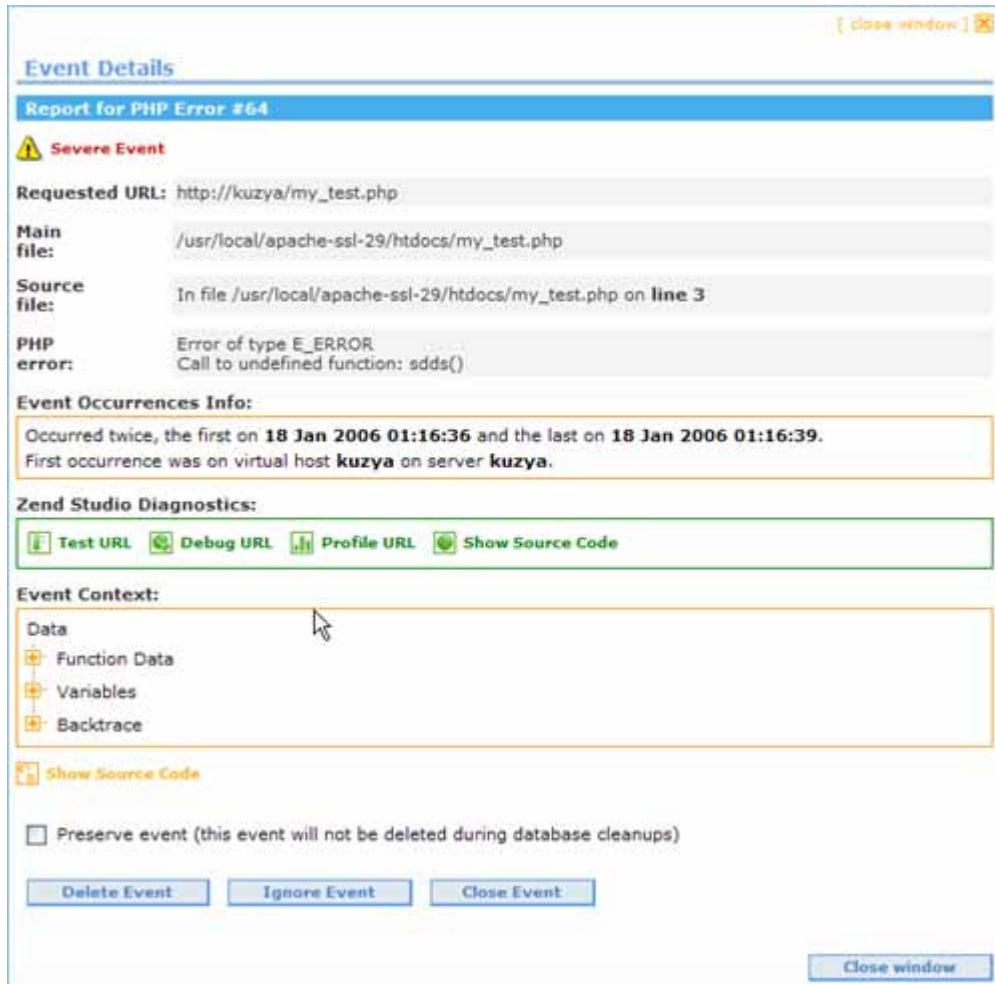


Figure: 29 - A PHP Event Details Screen

Event Details include five sections:

1. General Information
2. Event Occurrence Info
3. Zend Studio Diagnostics
4. Event Context
 - i. Function Data

- ii. Variables
- iii. Backtrace
- iv. Included Files
- v. Show Source Code

5. Event Administration

The information included in Event Details varies according to event type. For example: PHP Error Event Details include different information than a Slow Script Execution Event Details, simply because these events require different information to perform diagnostic analysis.

For example: A PHP Error will include in the General Information, the error's text and in the Event Context, the Function's Data. However, in a Slow Script Execution Error there is no need for the error text or the function's data and there will be information on how long the script ran for, included files and the load average at the time of the event.

Note:

If there is not relevant data to display in an Event Details screen, the section will not be included rather than appearing empty.

General Information

The general information section of the Event Details provides basic information about the event (depending on the event's type) as follows:

Report for PHP Error #64	
⚠ Severe Event	
Requested URL:	http://kuzya/my_test.php
Main file:	/usr/local/apache-ssl-29/htdocs/my_test.php
Source file:	In file /usr/local/apache-ssl-29/htdocs/my_test.php on line 3
PHP error:	Error of type E_ERROR Call to undefined function: sdds()

Figure: 30 - Event Details - General Information

- **Title** - The top of the Event Details, displays the event that generated the Error and the Event ID (Error #). The Event ID can later be used, to locate the event in the Event List or the Console in the filter's Find section.
- **Severity Level** - An additional notification is added to severe events.
- **Event Status** - The Status of the event is indicated for all statuses except Opened.
- **Requested URL** - The requested URL
- **Main Filename** - The URL's main file
- **Source Info** - The path to the Source File and line in the code that triggered the event.

Note:

Especially with code related errors, this information can provide an immediate indication to the source of the error in the code.

Trigger Value The script's trigger value (runtime,output size,memory consumption etc.)

- **CPU Load** - The CPU load when the event occurred.
- **Zend Error/ Error Description** - Shows the event's error text (for code related errors) and the \$type of Custom Events
- **Aggregate Hint** - Shows the Aggregate Hint for this event
- **Associate Zend Error** - Event Adds a link to an associated Zend error event.

Event Occurrence Info

To prevent an event from being continually reported for the same or similar event, PHP Intelligence enforces Aggregation Rules. These rules are based on a set of predetermined algorithms that determine which events are identical or are similar, to the extent they can be reported as a single Event. Aggregation Rules are also aware of events that occur on nodes belonging to a cluster – Groups and not just occurrences on a single server (node).

Aggregation information is displayed in event details to identify the number of times the event occurred and in case of Groups the information is expanded to include the servers on which the event occurred and the number of occurrences.

Event Occurrence Info:
Occurred twice, the first on **18 Jan 2006 01:16:36** and the last on **18 Jan 2006 01:16:39**.
First occurrence was on virtual host **kuzya** on server **kuzya**.

Figure: 31 - Event Details - Event Occurrence Info

The Event Occurrence Info section shows, the total number of occurrences, the time and date of the first and last occurrence and the first server (and vhost) on which the event occurred.

In cases where an event occurred on several servers belonging to the same group, an additional expandable list is added. This list displays occurrences per server, i.e. the server's name, and total number of times the event occurred on the server.

Zend Studio Diagnostics

The Zend Studio Diagnostics section shows the advanced diagnostic options that can be performed on the event's data.

These diagnostic options reconstruct the precise conditions that generated the event by recreating the request with the same parameters that were in the original request (Information such as: GET/POST/COOKIE/etc.).

Note:

The recreation process will not create an additional event.



Figure: 32 - Event Details - Event Occurrence Info

The diagnostic options that can be applied to event information are as follows:

- **Test URL** - Loads the exact same URL from the event with the exact same parameters (GET/POST/COOKIE/HTTP HEADERS/etc.) and shows the script's output in the browser.

Note:

The Test URL option does not require the Integration with Zend Studio.

- **Debug URL** - Initiates a Debug session of this URL in the Zend Studio.
- **Profile URL** - Profiles the URL, using the Zend Studio Profiler with the same parameters (GET/POST/COOKIE/HTTP HEADERS/etc).
- **Show Source Code** - Opens the file where the event occurred in Zend Studio. This option provides a means for editing files and implementing changes to multiple servers using Zend Studio.

Important: Debug URL, Profile URL and Show Source Code can be activated when the following conditions are met:

1. Zend Debugger is installed on the server where the event occurred.
2. Zend Studio (ZDE) is open.
3. Zend Core is configured to allow Communication between Zend Platform and Zend Studio.

Event Context

Event context includes relevant information available at the time of the occurrence. This information varies according to the type of event generated.

There are four main Event context categories:

1. [Function Data](#)
2. [Variables](#)
3. [Backtrace](#)
4. [Included Files](#)



Figure: 33 - Event Details - Event Context

Function Data

Function Data is only added to function related Event Details. This addition shows the function's name and parameters at the time the event occurred.

The function always appears as a link. This link directs to the function's description in the online PHP manual at: <http://www zend com/manual>.

Note:

The link to the PHP manual also appears for user-defined functions. These functions naturally, will not be found in the PHP manual however, it is a good indication as to which functions are PHP functions and which are user-defined.

Variables

The information included in the Variables section, includes all variables data saved when the event occurred, such as: GET, POST, COOKIE, SERVER, etc.

The GET, POST, COOKIE and SERVER sections will always be displayed even if they are empty. This indicates that there was no available data at the time the event occurred.

Users may choose to change the type of information collected and displayed in an event (Change Event Details).

Backtrace

Backtrace only appears in function related events. The listed functions are the functions that lead to the actual function (occurrence) that triggered the error.

Functions are listed in chronological order from the most recent to the first function that was called.

There are two options for viewing Backtraced functions, in a pop-up screen or in Zend Studio.

- Shows the function call in a pop-up screen.
- Shows the function call in Zend Studio.

Included Files

The Included Files section only appears in slow script error events. The files listed are all the files that are included in the PHP script that caused the error to occur.

Show Source Code

Shows event data in the Event details in the form of an expandable text field. This option opens the file in the section of code where the event occurred.

To view the source code in Event Details:

Press Show Source Code to expand the text area.

Event Administration

The Event Administration section includes all the actions that can be applied to an event see Controlling Information Displayed in an Event, to learn how to disable these options for certain users.

Preserve event (this event will not be deleted during database cleanups)

[Delete Event](#)

[Ignore Event](#)

[Close Event](#)

Figure: 34 - Event Details - Event Administration Actions

The applicable actions (Buttons) are as follows:

- **Preserve Event** - Keeps the event in the database even during database cleanups.
- **Delete Event** - Deletes the event (i.e., removes the event entirely from the database.)
- **Ignore Event** - Ignores future instances of this event (i.e., changes the event status to ignore). Therefore, if the same event occurs again, Platform will not open a new event.
- **Close Event** - Closes the event (i.e., changes the event status to closed). Therefore, if this event occurs again, Platform will open a new event.
- **Reopen Event** -Changes the event's status from closed to open.

Controlling Information Displayed in an Event

Zend Platform's User Management settings (Zend Central | User Management), can be utilized to set restrictions per User Group. These restrictions can control permissions to view event information and prevent certain User Groups from changing Event Details status.

The following restrictions can be applied to Event Details information:

- Delete an event
- Ignore an event
- Close an event
- Reopen an event
- See the event internal data in the Event Details
- See the event source code in the Event Details
- Use the Zend Studio Diagnostics in the Event Details

These limitations can prove to be especially useful for the organization. For example: when working in collaboration with external organizations that should not be permitted to view information such as the source code.

Another example of the Event Details restrictions is seen when implementing development lifecycle processes that require that certain groups be limited to the actions that they can do with an event such as closing or reopening.

Note:

To find out how to create a User Group, go to: "Configuring Users and User Permissions.

Customizing Events

Zend Platform provides several ways for customizing events in order to facilitate different requirements.

- **Custom events** – for generating a User Defined event that is not based on specific PHP Intelligence, Event Triggers.
- **Event Callbacks** – for adding user defined information to Event Details.
- **Aggregation API** – for setting different events to be aggregated with other events.

Custom Events

Custom events are a unique type of event that is provided for Zend Platform users in order to initiate events in their scripts. This type of event is different than other event types in that it allows controlling event generation as opposed to the other events that are triggered by a certain occurrence.

Custom events are used to generate an event whenever the API function `monitor_custom_event()` is called from the PHP script.

Description:

This event type enables the generation of an event on occurrences that are not necessarily built-in Zend Platform events (error and performance issues). Custom events are used whenever you decide that it is significant to generate an event in a certain situation. Each event type is given a name for easy identification (`$type`).

Function Usage:

```
void monitor_custom_event(string $class, string $text[, integer $severe, mixed $user_data])
```

Parameters:

- **\$class** – helps to define several types of custom events. This description will be showed in the Event List and in the Event Details.
- **\$text** - error text used to describe the reason for the event. This text will appear in the Event Details.
- **\$severe** - the severity level of the triggered event, default value is Severe.
- **\$user_data** - adds a PHP variable that will be viewed in the Event Details (in Event Context-> Variables->User Defined). This forms the stored Event Context (similar to the information obtained in a PHP error event).

Aggregation takes place for these events when, two events occur in the same place and have the same `$class $text $sever(ity)`

When viewing these events in PHP Intelligence | Event List, they can be filtered by the Custom Events category.

Note:

Event Actions defined for these events should be set to “send to URL” rather than “sending by e-mail” as there is only one definition for these events and Event Details sent to a URL can be easily forwarded elsewhere. This is to prevent the overloading of e-mail. If we use the e-mail action, for every custom event, e-mail will be sent, and there can be many classes of custom events. However if the URL action is used, a script can be used to identify the event’s class and different behaviors can be implemented according to class.

(To find out how to leverage Event Details information sent to URLs go to: Tutorial - Integrating Existing and Legacy Applications)

Event Callbacks

The event callback mechanism is used for viewing additional information about local variables in order to investigate what happened when an event was generated. The additional information is displayed in the Event Details.

Event Callbacks are created by extending information already provided in Event Details to provide an audit trail for problem conclusion.

Register and Un-register User Event Handlers

The event callback mechanism uses the following API functions:

- register_event_handler
- unregister_event_handler

Register Event Handler

To register a user function as an event handler, the following API can be implemented:

```
register_event_handler($event_handler_func  
[ [, $handler_register_name], $event_type_mask ])
```

Parameters:

- \$event_handler_func - The first argument is a callback function that will be called when the event is triggered. Object methods may also be statically invoked using this function, by passing the array (\$objectname, \$methodname) to the function parameter.
- \$handler_register_name - The second argument is optional and represents the name under which the function is registered. If no name is specified, the function will be registered under its own name.
- \$event_type_mask - The third parameter is an optional mask of event types on which the handler should be called. The default setting is MONITOR_EVENT_ALL.

When a monitor event is triggered, all the user event handlers are called and the return value from the handler is saved in an array keyed by the name under which the event handler was registered. The event handlers' results array is saved in the script_runs table.

Notes:

The first parameter is the name of the called function and it has to be a user-defined function. Built-in functions will not work with this API.

This function can get as a parameter the event type by which it was called.

If there is a PHP function register_error_handler in the JavaScript, events will not be reported. To report events call the function monitor_pass_events in the error handler.

Global Events should not be changed under any circumstances as they may produce unpredictable results.

Un-register Event Handler

The un-register event handler allows you to un-register an event handler. The API returns false if it cannot find a handler registered under the supplied name.

```
unregister_event_handler($register_event_handler)
```

Note:

Do not add the `unregister_event_handler` function to the end of scripts if you need to generate memory and script execution Events. These event types generate the event only after the script is executed and if `unregister_event_handler` is added it will stop the event from being generated.

The event types that should not include `unregister_event_handler` are as follows:

- Slow Script Execution Absolute
- Slow Script Execution Relative
- Inconsistent Output Size

Aggregate Hints

"monitor_set_aggregation_hint (page name)"

This API is a global variable that can be set anywhere and in any hierarchy. The purpose of this API is to incorporate locations of occurrences in the script.

This API is used when there are events that require the location in the script for diagnosing the reason behind the event occurring.

For example:

Global Events require the application that generated the event. Adding the Hint API can assist in the identification process.

Event Aggregation Rules

Event aggregation rules determine which events are aggregated into a single Event Detail.

There are four types of checks depending on the event type:

1. Database Error
2. Zend Error (PHP Error)
3. Function Error
4. Query Error

The Collector checks to see if these events occurred in the same source file based on the Line, Function and Hint. This is why it is important to used Hints if this level of separation is necessary.

Note:

To read more about the event aggregation mechanism go to: Appendix D - Event Aggregation Mechanism.

Database Maintenance

Once the cause of the event has been fixed, we can decide what to do with the event: Preserve, Ignore or Delete. If no actions are done to an event, it will be automatically deleted from the database.

Apart from closing events, other additional advantages can be obtained by customizing user permissions. Granting different users separate authorizations, by configuring different user permissions can facilitate different organizational requirements such as enforcing responsibilities and work structures. For example: by providing "Read Only" authorization to people who only need to see event details and granting authorization to Close events only to those who should close events (such as managers or team leaders) we can create and maintain a structured working environment.

The Event List provides four options for handling events in the system:

Close Event

Closes the event (i.e., changes the event status to closed). Therefore, if this event occurs again, Platform will open a new event.

Ignore Event

Ignores future instances of this event (i.e., changes the event status to ignore). Therefore, if the same event occurs again, Platform will not open a new event.

Delete Event

Deletes the event (i.e., removes the event entirely from the database.)

Manual Override

Users can also manually change the status of an event by clicking on the event in the Event List and changing its status. This method is helpful, for example, if you want to "un-ignore" an ignored event and restore it to the main screen.

This completes the PHP Intelligence chapter of the User Guide.

In this section we have described the Problem Resolution Lifecycle and how it can be implemented in an organization's environment.

Chapter 4 - Performance

IN THIS CHAPTER...

OVERVIEW

CODE ACCELERATION

CODE OPTIMIZATION

FILE COMPRESSION

DOWNLOAD SERVER

PERFORMANCE LIFECYCLE

BENCHMARKING - SITE ANALYSIS

EVENT TRIGGER SETTING AND ANALYSIS

PERFORMANCE OPTIMIZATION TOOLS

WHEN TO APPLY OPTIMIZATION TOOLS

DYNAMIC CONTENT CACHING

CODE ACCELERATION

CODE COMPRESSION

ZEND OPTIMIZER

ZEND DOWNLOAD SERVER (ZDS)

The Zend Performance module provides a collection of comprehensive tools for enhancing PHP Web applications and Server performance in Enterprises.

Using Performance Provides:

- Increased server throughput, with less hardware
- Improved user-experience, with faster response time and download time
- Reduced stress on production database servers and http servers
- Reduced costs on new hardware purchases and IT maintenance operations
- Better utilization of existing hardware resources and capacity

Overview

The Zend Performance module consists of several components for providing server performance optimization:

- Code Acceleration
- Code Optimization
- Dynamic Content Caching
- File Compression
- Download Server
- Performance Tests

Code Acceleration

Code Acceleration begins from the first moment Zend Platform is installed. The Zend acceleration component performs a pre-compilation of your PHP scripts, eliminating the lag time and interpreter time involved in script parsing. During compilation, the code is also

optimized, resulting in even faster execution time.

How it works: Server-side pre-compilation generates persistent bytecode. Modified scripts are automatically detected. Compiled scripts are optimized using advanced code optimization methods.

Code Optimization

Code optimization begins from the first moment Zend Platform is installed. The Zend optimization component performs several passes, each pass searches for specific points in the PHP code that are known to have a negative affect on performance and changes them for faster execution.

Dynamic Content Caching

Dynamic Content Caching dramatically reduces the number of times your server must run complex scripts, execute resource-intensive database queries, or call external web services.

How it works: Server-side caching eliminates the need to return to databases, duplicate processes or re-build a web page for each page access. Cached versions of any URL can be maintained for any amount of time that you determine. Fully configurable parameters determine what to cache and based on which conditions. No code-level modifications are required. Moreover, you can use the PHP API for partial and conditional caching of parts of script functionality.

File Compression

File Compression increases the end-user download speed and decreases the workload on your http server. Better than any other compression option due to integration with Dynamic Content Caching eliminating the time it takes to run the compression.

How it works: Specific browser capabilities are auto-detected. If browser supports gzip format, the results are compressed prior to returning to the user. Both the original and the compressed version are cached and reused, depending on browser capability and cache lifetime.

*Download Server**

The Zend Download Server is a PHP (Zend Engine) plug-in which efficiently deals with serving large downloads such as videos e.g. .mpeg files, binary products such as .exe and .msi files, and any other large files which are served over the HTTP protocol.

How it works: The Zend Download Server supports two modes of operation (both of which can be used together or separately according to your needs):

- **Manual mode** - The download is initiated by a PHP script using one simple PHP API function call. Not only does it allow you to serve files which aren't under your web server's document root but also it allows you to run logic such as access restriction checks before the download is started.
- **Transparent mode** - In your web server's configuration file you map the files you want to be sent through the efficient downloading mechanism to PHP, and the Zend Download Server will jump into action automatically and serve them.

*The Download Server is currently not applicable for Windows Operating Systems.

Performance Lifecycle

Maintaining Web applications at optimal performance levels is a necessary requirement for ensuring customer satisfaction and organizational efficiency. Zend Platform's Performance module provides tools for optimizing Web application performance by employing a detailed

performance enhancement method – the Performance Lifecycle. The Performance Lifecycle is a process of calibrating Zend Platform to provide an optimal performance boost to business critical Web applications.

Deploying Zend Platform in organizations will improve the overall performance of Web applications by:

- Enhancing code and download performance.
- Employing full and partial page caching capabilities.
- Preserving memory consumption through file compression.

The use of Zend Platform Performance tools in development and production environments provides a means for testing and maintaining Web application performance.

The following illustration displays the three stages of the Performance Lifecycle.

Zend Platform Performance Lifecycle



Figure: 35 - Performance Lifecycle

The Zend Platform Performance Lifecycle is an iterative cycle for analyzing Web application performance. The purpose of this cycle is to identify areas that require Zend Platform calibration and areas that require PHP code optimization.

The Performance Lifecycle Baselines (stages) are as follows:

Zend Baseline

The first Baseline is the Zend Baseline. This baseline measures performance based on Zend Platform's default parameters that are immediately activated upon installation. The default parameters are, Event Trigger settings, Code Optimization and Code Acceleration. Once Zend Platform is installed, these components automatically begin to work on the PHP code. The result is an immediate improvement to the Web application and initial PHP Intelligence event generation (based on default Event Trigger settings).

The purpose of the Zend Baseline is to evaluate overall performance in relation to the Zend Platform defaults. This information is used as an initial starting point for subsequent calibration and optimizations.

Note:

In the Zend Baseline stage, it is common to experience abnormal event generation behavior (too many or too little events generated). This is a normal part of the initial calibration stage, necessary for identifying how to adjust performance settings to obtain optimal Web application performance.

Site Baseline

The second Baseline is the actual calibration process. Based on information collected and observed in the first Baseline, the performance settings can be calibrated to suit each organization's specific Web application. The Site Baseline enables one to obtain insight into the overall performance of the Web application. Once the Site Baseline is established by configuring Event Triggers these events can be further analyzed to evaluate the mode of action required to optimize the Web application's performance. At this point it is recommended to perform a Site Analysis to benchmark the Web application. The Benchmark information provides an initial indication of the Web application's current performance before applying the additional performance tools. This will provide a point of comparison to view improvements that occur after subsequent optimization is done with Zend Platform.

After the Site analysis, the PHP code can be optimized. Optimization is obtained by implementing performance tools to areas in the PHP that exceed the Site Baseline settings (still generate events).

There are four possible choices for adding performance features to PHP code:

1. Full page caching
2. Partial page caching
3. Compression
4. Blacklist files or directories

Completing optimization of the Site Baseline brings us to the Optimized Baseline.

Optimized Baseline

The Optimized Baseline represents the stage where the Web application is optimized and Zend Platform is calibrated with the Web application. From this stable stage all that is left to do is to let Zend Platform perform regular Production Monitoring.

Note:

When the Web application is re-deployed or changes are made, this process should be repeated from the Site Baseline stage in order to reestablish the Optimized Baseline.

Now that we have established what the performance lifecycle does and the tools it comprises. The next step is to see how to implement the performance lifecycle. At the end of this guide you will find a Performance Lifecycle Check List that details the steps to establishing an Optimized Baseline for Web applications. The next chapter "Implementing the Performance Lifecycle" details each of these steps.

Implementing the Performance Lifecycle

The following section provides a detailed instructional overview of performance optimization features and components for implementing the Performance Lifecycle.

Benchmarking - Site Analysis

Site Analysis enables to obtain insight into the overall performance of Web applications. Benchmark information provides an initial indication of the Web application's current performance. This information can be used as a starting point for observing the performance boost gained applying the performance tools. Benchmarking measures Web server performance and durability.

In Zend Platform, Benchmarking is achieved through the Testing screen (Performance | Testing).

This screen includes three options.

- **Test URL** – tests a single script, running Performance and Compression tests at the same time. The test results indicate the script improvements achieved by Code Acceleration, Dynamic Content Caching and File Compression.
- **Analyze Site** – tests performance for the entire Web application, running the Performance test separate from the Compression test. The test results indicate the overall script improvements achieved by Code Acceleration, Dynamic Content Caching and File Compression and the popularity of each file.
- **Test Download** – tests the efficiency of the Zend Download Server. This test is addressed in a separate section at the end of this chapter titled "Zend Download Server".

The above-mentioned tests analyze an entire site's performance or monitor a single script. The test results can be further used outside Zend Platform as they can be printed or sent by e-mail.

Notes:

Since testing may take a while to run, it is suggested that you choose only the most recently added files. You may select as many files as you wish; nonetheless this will increase the duration of the test.

Prior to running the Compression Test and in order to ensure accurate results, you may want to add query strings to the script path entries.

When running Performance Tests, query strings can only be added to cached scripts, to check for performance gain.

Analyze Site (Benchmark)

Benchmark Web applications:

To Benchmark Web applications:

Go to Performance | Testing and select the Test URL Tab.

To test a script, follow these steps:

1. Click Test URL and type the full path of the script. To select a previously tested URL, click Show History. By default, URLs are tested using GET variables defined in the query string.
2. Press Add variables to URL to add the variable Name and Value to test URLs using specific SESSION or COOKIE variables. Add the User Name and Password to test URLs that are restricted by HTTP Authentication.
3. To delete any variable from the list, click next the variable (To add specific SESSION variables to the test, make sure that your PHP is configured correctly to work with sessions. For example: if you use 'files' as your session.save_handler, confirm that the session.save_path is a valid path. If you use 'user' as your session.save_handler, you must prepend the file containing the user-level session storage functions).
4. To determine the Duration of test, specify the time in seconds (per script) in the Duration of test box.
5. Press Run.

Test Results

The Test Results screen depicts the Dynamic Content Caching Results, the Code Acceleration Results, and the File Compression Results.

- The Dynamic Content Caching Results compares between the Base script and the script after it has been cached, and calculates the total performance improvement accomplished due to caching.
- The Code Acceleration Results displays the performance gain. If the script is cached, acceleration does not improve performance acceleration is therefore not necessary
- The File Compression Results compares between the original file size and the compressed file size and calculates the savings in bytes and the improvement in percentage.

When running the Analyze Site test, you can choose to run the Performance test separately from the Compression test:

- **Performance Test results** - indicate the average improvement achieved by Code Acceleration, Content Caching and the average overall improvement achieved;
- **Compression Test results** - present the improvement accomplished due to the File Compression.

To Analyze a Site:

Go to: Performance | Testing and select the Analyze Site tab.

The following options are presented in the tab:

- Run Performance Test
- Run Compression Test
- Display Previous Test Results

Performance Test

1. Select the number of scripts to test and press Next.
2. The following screen lists the scripts selected. To change the number of the scripts to be tested, press Previous. To continue, press Next
3. The Site Analysis Report appears indicating the Performance Gain and Popularity Rank of the scripts.

N/A Test Results

N/A test results can be caused by various reasons.

To check the cause of a N/A test result:

Place the cursor on top of the N/A and the cause of the problem appears in a Tooltip. N/A can be caused when a script cannot be accessed or if the access time to the script is greater than the test duration (3 sec). If this is the case, test the script separately using the Test URL option.

After analyzing the test results you may wish to modify the caching status of the scripts and re-run the test.

To re-run the test:

Press Edit next to the script you wish to modify, change the caching conditions and save the new settings. If the message appears, the test results are incorrect since the changes you made did not take effect. In this case, restart the server and simulate a typical user session to get valid results.

Compression Test

The Compression test analyzes download time improvement of the popular scripts as the result of Compression.

1. Press Run Compression Test.
2. Choose the number of scripts to test and press Next.
3. To ensure accurate results for the scripts, add query strings in the Script Path entries. For example: /site/example.php?var1=value1&var2=value2).
4. Press Run to run the test.

The Site Analysis Report screen details the results for the Compression test. The report shows the Original script size, along with the Compressed size and the Compression Improvement rate and the actual compression functionality is not affected.

Note:

If one or more scripts failed the test, an indicative message appears on the screen. The Compression Test failed since Platform Performance cannot resolve the URL from some script paths or the URL cannot be accessed. Note that scripts defined on a virtual host or a symbolic link can cause the test to fail.

Display Previous Test Results

The last Test Results summary is displayed in the Testing environment.

To display last Test Reports, click the Show Last Performance Test Report or Show Last Compression Test Report button. The displayed Test Report will be updated as soon as you run another test.

Note:

Test results can also be sent by e-mail and stored outside Zend Platform

Event Trigger Settings and Analysis

Initially Zend Platform Event settings are based on predefined default parameters. As such these settings require calibration to suit specific environments. The anticipated results are Event Details generated based on default triggers. Running Zend Platform for the first time will most probably cause abnormal event generation behavior (to many or to little events).

This behavior is attributed to two causes:

1. The Default triggers need to be calibrated.
2. The PHP needs to undergo additional performance Optimization.

Before making any adjustments to the PHP code that may turn out to be unnecessary. The first action that should be performed is to calibrate PHP Intelligence.

Calibrating Event Triggers for Performance Optimization

Calibrating PHP Intelligence is the process of adjusting Event Triggers to accommodate a specific Web application.

Event Triggers are calibrated from PHP Intelligence | Event Triggers.

Note:

Rules are configured on a selected server and can be subsequently applied to additional nodes with the Clone Wizard. This option should be used in node cluster environments.

When calibrating Event Triggers for performance optimization, the following performance-related event types should be addressed:

- **Slow Script Execution (Absolute and Relative)** – generates an event when script execution exceeds defined limits
- **Slow Query Execution** – generates an event whenever database related functions exceed the threshold defined in the event.
- **Slow Function Execution** - Generates an event when a specified PHP function's execution time exceeds the threshold defined in the event.
- **Excess Memory Usage (Absolute and Relative)** - Generates an event when memory use for PHP script execution is above or below average.

These event types are performance related indicators. Each one of these events should be configured to generate an Event according to your environments performance requirements. Events generated following calibration indicate that certain areas of the application are not performing according to your requirements and need further investigation.

Investigating Performance Related Events

Performance related events are investigated when events continue to occur after initial calibration. The contents of Event Details provide in-depth diagnostic information and tools for investigating the occurrence. A generated event does not necessarily indicate a problem with the PHP Code, it can also indicate that the Event Trigger settings need to be adjusted or the PHP code should be reevaluated.

The most effective performance diagnostic tool is the Zend Studio Profiler. The Zend Studio Profiler is a Zend Studio component that can be employed on Event Detail information through Zend Platform's integration with Zend Studio.

Profiling PHP Code With Zend Studio

Running the Zend Studio Profiler on PHP code provides time-related snapshot of the Code's overall performance. Profiling uses the Zend Studio IDE tools, to analyze PHP code. When profiling, the event's information is transferred from Zend Platform to Zend Studio. This information includes all the information necessary to precisely recreate of the actual occurrence that generated the event. The Zend Studio Profiler is so accurate that this process is paramount to running the profiler when the event originally occurred. Profiler information is generated by, placing timers within the code and running them over and over. The profiling tool is able to build a "profile" of how fast or slow specific areas of the application will run.

The Profiler is activated through Event Details from the Zend Studio Diagnostics section by selecting the option, Profile URL.



Figure: 36 - Event Details - Zend Studio Diagnostics, Profile URL

The Profiling process takes place in the Zend Studio IDE and automatically opens the profiler results.

Note:

Zend Studio has to be installed and running to profile code.

Understanding Profiler Results

Based on the information provided with the profiler, developers can identify the cause for the performance problem and implement changes to the code accordingly.

The Profiler user interface contains 3 tabs:

- **Profiler Information** - provides general information on the profiling duration and date, number of files constructing the requested URL and more. In addition, it displays a Time Division Pie Chart for the files in the URL.
- **Function Statistics** - provides you with the list of files constructing the URL and detailed information on functions in the files.
- **Call Trace** - provides a hierarchical display of functions according to process order, enabling you to jump to the function, view the function call, function declaration, details and more.

Note:

Additional information about the Studio Profiler can be found in the Zend Studio Online Help.

To perform in-depth examinations of slow code or functions the Zend Studio Debugger can be used to debug information in the occurrence's relevant context.

The Debugger is also activated through Event Details from the Zend Studio Diagnostics section by selecting the option, Debug URL.



Figure: 37 - Event Details - Zend Studio Diagnostics, Profile URL

This completes the description of the investigation and diagnostics tools that can be used to identify performance bottlenecks in the code. The next step is to see what performance tools can be applied to optimize the Web application's performance.

Performance Optimization Tools

Performance optimization tools are used once it is apparent the code is performing as it should, and the Event Triggers are calibrated.

From this stage on the performance optimization tools can be applied to further enhance Web application performance.

There are several levels of performance optimization that can be applied to files: Caching (full page), Acceleration and Compression.

Applying these three settings to your PHP code provides optimal performance boost.

Optimization Tools:

Caching, Acceleration and Compression

The default settings for these optimization tools are as follows:

- **Caching** - Default setting Off
Description - Runs code and saves the output on the server
- **Acceleration** - Default setting On
Description - Compiles the Code and saves the compiled code on the server
- **Compression** - Default setting Off
Description - Saves a compressed version of the code on the server

To apply these optimization tools to all of the PHP files on the server caching and compression must be activated.

To activate/disable Caching, Compression or Acceleration, go to Performance | Settings and activate the enabling options.

When to Apply Optimization Tools

Now that we have established how to apply the optimization tools to "All Files" it is important to state that there are different circumstances that require disabling one or more of these features for select directories or files and in some cases altogether. The following section describes the possible optimization alternatives that should be considered when applying optimization tools.

Content Caching (Dynamic)

Dynamic Content Caching is the process of running code once and saving the output on the server for reuse in a set time frame (Cache Lifetime). Each time the code is requested, performance is improved by using the already run output instead of generating the same output each time.

When Should Files be Cached?

Files should be cached when their content is stable and does not require frequent changes.

When Not to Cache Files?

Caching is not recommended for files that have constantly changing output. For example: clocks, timers and database queries. (See Caching Alternatives to find out how to Partial Page Cache).

How to Cache Files

Caching by default is enabled (set to "ON"). In order to view/change the setting:

1. Click the Performance tab.
2. Pick a server to configure.
3. Click the Settings tab.

The dynamic caching enabled settings status will be displayed. Next, define the default caching settings. These settings are applied to all cached files.

To prevent unnecessary memory use, caching has to be actively applied to either a selected file or directory.

The performance module provides two options for caching files:

- Through the File view
- Through the Performance Test Report

Caching with the File View

Caching can be applied to single files or do entire directories. Go to Performance | File View and choose one of the following options:

1. Apply Caching to a single file
2. Apply Caching to an entire directory.

Specific caching settings, given to files and directories, override the main settings defined in: Performance | Settings.

Caching with the Performance Test Reports

Performance Test Reports provide site analysis information in terms of performance gain and popularity. The information included in these reports provides a strong basis for evaluating if a file should be cached.

Site Analysis Report - Performance Test For http://gollum:80		[23-Jan-06, 17:07]	
Edit	Script Path	Performance Gain	Popularity Rank
Edit	/home/root/site/zde_pofiler_bug.php	Not Improved	98.01% (2608 hits)
Edit	/home/root/site/ZendPlatform_2_1_2/infra/gui/styles.php	x3.26	0.41% (11 hits)
Edit	/home/root/site/ZendPlatform_2_1_2/zps/run.php	x25.75	0.19% (5 hits)
Edit	/home/root/ZendModules/ZendPlatformGUI/zps/run.php	x3.00	0.19% (5 hits)
Edit	/home/root/ZendModules/ZendPlatformGUI/zps/site_stats.php	x3.99	0.15% (4 hits)
Edit	/home/root/site/ZendPlatform_2_1_2/zps/gui/zps_styles.php	x1.94	0.15% (4 hits)
Edit	/home/root/site/ZendPlatform_2_1_2/zps/site_stats.php	x23.41	0.15% (4 hits)
Edit	/home/root/ZendModules/ZendPlatformGUI/zps/tabs.php	x1.62	0.08% (2 hits)
Edit	/home/root/ZendModules/ZendPlatformGUI/zps/uritest.php	Not Improved	0.08% (2 hits)
Edit	/home/root/site/ZendPlatform_2_1_2/zps/uritest.php	x23.36	0.08% (2 hits)
Average improvement for accelerated files: x0.95		Average improvement for cached files: x1.00	Average overall improvement: x1.12
 Cached  Accelerated			

Figure: 38 - Performance Test Report

This screen shows if a file is cached and provides an option to cache a selected file from the list.

To Cache a file from the Site Analysis Report:

1. Go to Performance | Testing and select the Analyze Site Tab.
2. Run a performance test (or view the last performance test)
3. Go to the test results and press Edit. This will open the “Define Caching Conditions” dialog.

Caching Alternatives

Web pages that contain sections that continuously change can also be cached. This partial page caching solution can be accomplished through, applying caching APIs to portions of code that do not change. Partial Page Caching provides an intermediate solution for providing a partial performance boost that sustains the accuracy of changing content. To find out more about “Partial Page Caching” go to: “Tutorial - Partial and Preemptive Page Caching,”

Note:

Dynamic Content Caching can be deactivated from Performance | Settings and changing Dynamic Caching Enabled to Off. This will remove all Dynamic Content Caching settings from the files on the server. However, Partial Page Caching will not be affected. Partial Page Caching can only be disabled by, removing the Caching APIs from the code.

Code Acceleration

Code Acceleration is the process of gaining a performance boost by eliminating the code compilation time. Once PHP code is compiled for the first time, it is saved in the server's memory. Each time the code is called, the pre-compiled version is used instead of incurring a compilation lag each time the code is used.

Note:

Acceleration should not be confused with Caching. Acceleration saves the compiled script in the server's memory whereas Caching saves the script's output in the server's memory.

When Should Files be Accelerated

The general recommendation is to always use Code Acceleration to boost Web application performance. Therefore, the default setting for Acceleration is set to: On.

When Not to Accelerate (Blacklist)

There are some instances where it is preferable to disable acceleration for select files. Acceleration is disabled by means of a Blacklist. Files should be added to the blacklist under the following conditions:

- Directories containing files that are larger than the Accelerators memory allocation or containing more files than the allocated quantity of files.
- Large files that have high memory consumption
- Files that have long execution time (makes the compilation save irrelevant).

Increasing Accelerator Memory Allocation

The alternative to blacklisting files is to increase the Accelerator memory allocation. The accelerator settings can be changed to increase allocated memory and the maximum quantity of files that can be accelerated. This alternative depends on the amount of memory available for allocation to the Accelerator.

When the Zend Accelerator is disabled, only cached scripts are tested.

To enable the Zend Accelerator, set the 'zend_accelerator.enabled' directive in the php.ini file to 'On'.

To change Accelerator memory allocation:

1. Go to Performance | File View
2. In the Code Acceleration section:
 - a. Increase the Accelerator Memory
 - b. Increase the Maximum Accelerated Files (default 2000)

Note:

If the memory fills up quickly, especially if there is only a small amount of Accelerated files. Increase the memory allocation or blacklist the file. Files exceeding allocated memory or quantity will not be accelerated.

Accelerator Duplicate Functions Fix

Some PHP code produces different opcodes for different situations, function defined or not. This causes a discrepancy for the accelerator in situations where the accelerator caches one version, and then a different situation occurs that requires a different function. If not addressed the script would just cease to work and raise a "duplicate functions" error.

To maintain proper performance in situations like these the zend_accelerator.dups_fix parameter should be activated. This parameter shuts down the Zend Accelerator's duplicate function check, so that the errors will not occur.

This parameter belongs to The Zend Accelerator settings in the PHP Settings screen (Zend Central | Console | Configure PHP Settings).

Reset Accelerator

Programmatically resetting the Accelerator with accelerator_reset()

You can programmatically reset the Accelerator by calling the PHP function accelerator_reset() from within your PHP script.

Note:

Under certain circumstances, such as a busy server or complex PHP processes, it may take a few minutes to reset.

While Platform Performance Accelerator is being reset, a certain degree of server performance degradation takes place, since the accelerated scripts cannot be used during the reset period. The server will run as if Platform Performance had not been loaded.

Note:

Platform Performance Accelerator memory is also reset whenever the Web server is restarted.

Code Compression

Code compression is the process of using less bandwidth and increasing performance by compressing code before it is sent.

When to compress files

The default setting for file compression is Off. However, files should be compressed under the following conditions:

- When the Web application's users are accessing the Web application with various low bandwidths that can benefit from the extra performance gained by compression.
- When there is a large quantity of Cached files.

When not to compress files

There are some instances where it is preferable to deactivate compression for select files. Compression can be deactivated in several ways:

- Deactivate compression entirely – should be done if the server is set to handle compression to prevent compressing files twice and rendering them unusable.

- Setting compression to cached files only – should be done when there is a large quantity of cached files and the rest of the files do not require compression.
- Blacklist – selectively disable compression for files do not require compression such as pictures that are already compressed or small files that do not require compression.
- When using PHP's compression feature zlib.

Setting Code Compression

To change File Compression settings:

1. Go to Performance | File View
2. Select the appropriate Setting:
 - a. **None** – disables Compression
 - b. **Only Cached Files** – applies compression to cached files only
 - c. **All Files** – collectively applies compression to all files (The list of files can be viewed in Performance | File View).

Zend Optimizer

The Zend Optimizer is a passive performance component that runs within the Zend Platform Framework and automatically optimizes scripts. Zend Optimizer is also designed to detect and load files encoded with the Zend Encoder.

Note:

The Zend Optimizer's default setting is "On" which means the Optimizer will start to run as soon as Zend Platform is installed. The Optimizer component does not require any additional configurations.

If you do not plan to use the Zend Optimizer to load encoded files, you can slightly improve the Optimizer's performance by adding the `zend_optimizer.enable_loader = 0`. This disables the transparent auto-loading mechanism that is built into the Zend Optimizer.

To change `zend_optimizer.enable_loader` settings:

Go to Zend Central | Console | Configure PHP Settings and choose from the list of directives Zend | Optimizer.

Test Download

The Zend Download Server (ZDS) is a transparent process that runs in the background to service large downloads. The performance gain obtained by using the ZDS is measured with the Test Download option.

The Test Download option checks the efficiency of the Download Server. Test Download simulates multiple requests for a specified URL with and without the ZDS, thereby creating its own benchmark for comparison. This feature is currently not applicable for Windows Operating Systems.

To Run a Download Test:

Go to: Performance | Testing and choose the Test Download tab.

1. Enter the URL for Testing. (The default is a proprietary Zend PHP script which uses `zend_send_file()` to send a file.)

Note:

Testing very large files will take a very long time.

2. Enter the bandwidth limit you want to simulate for the clients. For a faster test, select a higher bandwidth.

Note:

Do not select full bandwidth, doing so will saturate your network card that will make the test irrelevant. The test tries to simulate a typical Internet server that has clients connected either by ISDN or DSL.

3. Enter the Max Clients value defined for your server. Insert the accurate value by checking the `MaxClients` directive in the `httpd.conf` file.

Note:

The ZDS installation tries to identify your `MaxClients` value via the `httpd.conf` file that is the GUI's default value, but this value can be changed after the installation, so it is important to verify that the value listed is correct.

4. Press Run to Run the Download Test.

Viewing Download Test Results

Once the tests have completed, you can view the test results both statistically and graphically.

To view the results of the most recent Download Test:

From the Test Download Tab press, " Show Last Download Test Report". The test report includes tables and graphs that display the Requests per Second and Number of Concurrent Requests for each test run.

Note:

The Download Test is a simulation. The most meaningful test results are obtained by running ZDS on the production server and monitoring the log file to see how many concurrent jobs the ZDS is handling.

This completes the Performance Lifecycle chapter of the User Guide. In this section we have described the Performance Lifecycle and how implement the Zend Performance tools in creating optimal performance levels in an organization's environment. Please refer to "Appendix C – Performance Lifecycle Check List" to read/print the Performance Lifecycle Check List that summarizes the performance optimization tasks.

Chapter 5 - Java Bridge

IN THIS CHAPTER...

JAVA BRIDGE TAB

ABOUT ZEND'S JAVA BRIDGE TECHNOLOGY

OPERATING AND CONFIGURING ZEND PLATFORM'S JAVA BRIDGE

THE JAVA BRIDGE USER INTERFACE

USABILITY ISSUES

COMMON TASKS

The Zend Platform PHP/Java Bridge is a PHP module that connects the PHP object system with the Java object system. It can be used to access Java based applications running in a Java application server.

Platform's Java Bridge offers significant performance and scalability advantages. Specifically, the memory consumption in the Platform PHP/Java Bridge is constant, regardless of the number of PHP sessions—unlike the equivalent solution, for example, in PHP 5.

The PHP/Java Bridge feature should interest three types of enterprises:

- Companies that have investments in J2EE application servers can take advantage of PHP's Web-enablement capabilities, while preserving the utility of their Java investment.
- PHP-centric companies that want to take advantage of J2EE services that are not present in scripting languages—specifically, PHP.
- Companies that are not highly invested in J2EE and legacy systems can take advantage of Platform's PHP/Java Bridge to interact with plain Java objects.

About Zend's Java Bridge Technology

Zend's Java Middleware module (JavaMW) provides PHP connectivity to Java. The API is analogous to the standard PHP Java API (<http://www zend com/manual/ref.java.php>), however the implementation is different. JavaMW uses a stand-alone Java server process, which allows it to efficiently process Java requests. It adds stability and reliability to the PHP/Java connection. Unlike a standard PHP/Java connector, it uses a single Java virtual machine for all the requests, which makes memory and processor requirements significantly more modest while improving scalability.

The diagram below illustrates Zend Platform's Java Bridge technology:

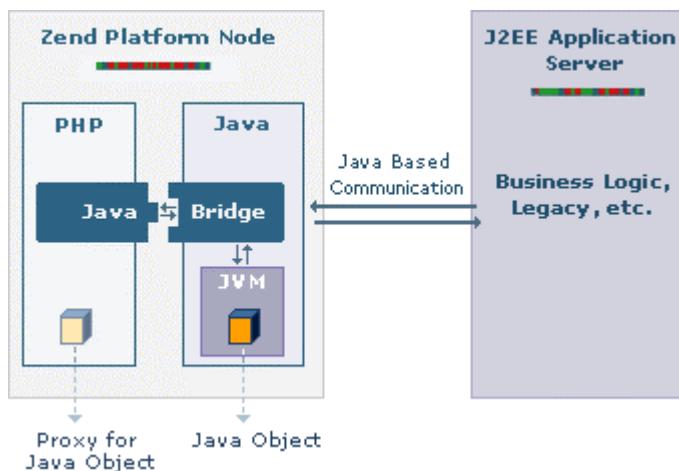


Figure: 39 - Java Bridge Process Level

The Java Bridge Process Level diagram illustrates the following:

Zend Platform Node

Zend Platform Nodes include two bridging components: the PHP-side Bridge and the Java-side Bridge. Zend Platform Nodes operate as follows:

1. A JVM (Java Virtual Machine) is installed first—before installing the Platform Node—on the machine that is to be set up with Zend Platform.
For the Java Bridge to function, you must install a compatible version of JVM. Platform will find the compatible version automatically. Supported versions are SUN J2SE 1.4 or SUN J2SE1.5 (J2SE 5).
2. Zend Platform then installs the two components required—the PHP-side and the Java-side—to create the Java Bridge.
3. A PHP application can call a Java object from any Java library that resides on the Node. For example, JVM can be downloaded with all its component libraries.

When a PHP application calls a Java object over the Java Bridge, a proxy for that object is created in PHP. In the diagram, the Java object is represented as a dark square; the proxy for that object in PHP is shown as a light square.

J2EE Application Server

The J2EE Application Server in its more advanced configuration, allows you to create a PHP/Java Bridge between a Zend Platform Node and an external J2EE Application Server. This type of configuration is typical of companies that have existing Java-based infrastructure. The J2EE Application Server operates as follow:

1. A PHP application can call a Java object from a Java library external to Zend Platform.
2. The Java-side Bridge component communicates with the J2EE Server. It finds objects in the J2EE Server, for example an EJB. The entire process is Java based.
3. The PHP application then calls the Java object over the Java Bridge created between the two Platform bridging components.
4. A proxy for that object is created in PHP. In the diagram, the Java object is represented as a dark square; the proxy for that object in PHP is shown as a light square.

The complete integration of Java and PHP is described in the following diagram:

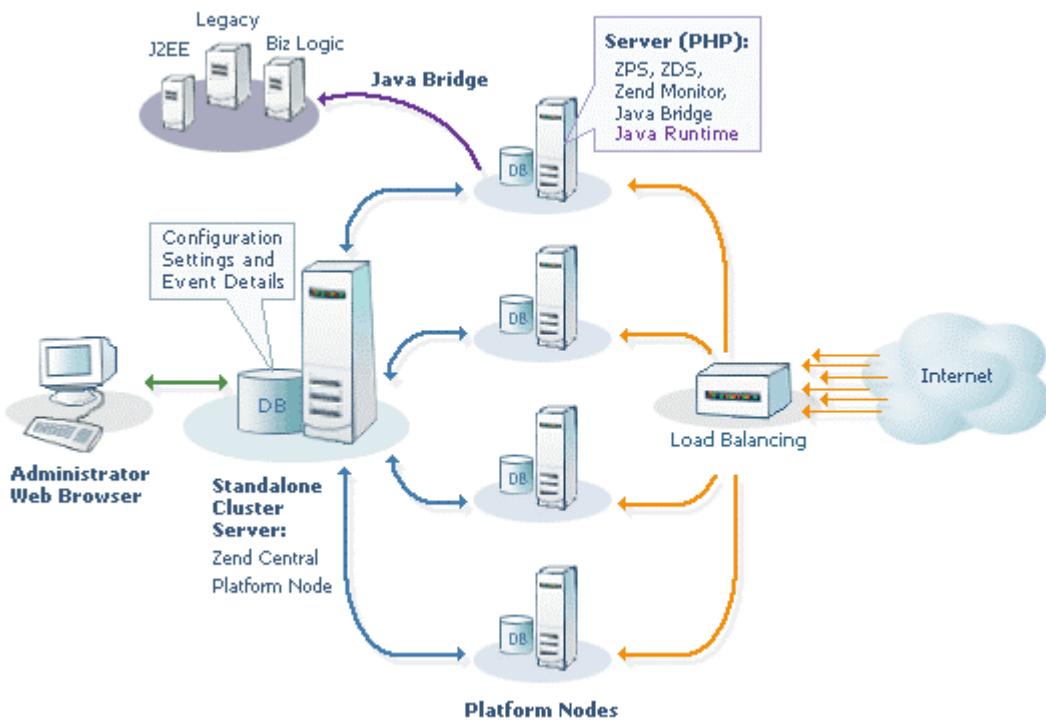


Figure: 40 - Java Bridge System Level

The Java Bridge System Level diagram illustrates the following about the network architecture:

- **Zend Platform Nodes** - In order for a Zend Platform Node to function as a Java Bridge, it must have a properly functioning Java installation. Once Java is installed, Platform installation installs the required components for the Java Bridge, some of which are implemented in Java.
- **J2EE Server** - The Java-based enterprise that adds Zend Platform will have its own application servers. A J2EE Server is shown in the diagram as part of the Front Office. It can communicate with any of the Platform Nodes that have Java installed on them and

which are defined in Java as legitimate accounts.

Added Value

Zend Platform's Java Bridge supports a PHP-Java integration that benefits enterprises on both the business and technical level.

Business Level Benefits:

- Companies with J2EE application servers can begin to realize the advantages PHP offers over other Web-enablement languages, including: shortened development time, shortened time-to-market, lower TCO (Total Cost of Ownership), etc.
- PHP-centric companies can take advantage of J2EE services that are not present in scripting languages.

Technical Level Benefits:

- Platform's PHP/Java Bridge provides the ability to interact with plain Java objects.
- Platform's Java Bridge operates without the overhead of a JVM for each Apache process.
- Platform's Java Bridge consumes a finite amount of memory, which is almost disproportional to the amount of activity that's going through it.

Operating and Configuring Zend Platform's Java Bridge

This section describes procedures for operating and configuring Zend Platform's Java Bridge.

Configuring Run-time:

For running JavaMW, the following command can be used:

```
java com.zend.javamw.JavaServer
```

For correct execution, classpath should include javamw.jar file in the directory where JavaMW is installed, e.g.:

```
Unix <install_dir>/bin/javamw.jar
```

```
Windows <install_dir>\bin\javamw.jar
```

Java Bridge

Java Status Page

The Java Bridge tab provides status information about the Java servers connected to the network. The information displayed in the Java Status Page, shows information about a selected server.

The active buttons on the Java Status Page are:

- **Stop** - Shuts down the Java Bridge daemon.
- **Start/Restart** - Restarts the Java Bridge daemon.
- **Refresh** - Refreshes the page for the selected server.
- **Help** - Opens the Online Help for the Platform Java Bridge.

The Java Status page includes information about:

- **Java Environment** - The Java Environment includes the Java Version, Java Vendor, OS Name, OS Version, Class Path and Java Home
- **Bridge Statistics** - Bridge Statistics information includes the Number of connections and Requests.

Locating an Existing Java Version

Zend Platform assumes that your existing Java is installed in the standard location:
/QOpenSys/usr/bin/java.

If this is not the case, to use the Java Bridge you must relocate your Java to:
/QOpenSys/usr/bin/java.

Working with the Java Bridge User Interface.

To view the Java Status Page for a selected server:

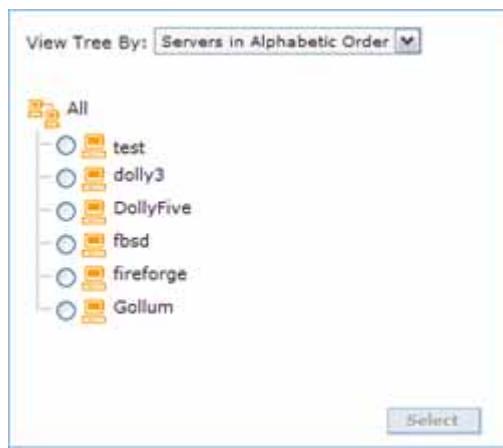


Figure: 41 - Select Server to Configure

1. Go to the Java Bridge Tab and the "Select Server to Configure" dialog opens.
2. Select a server from the list of servers in the Server Tree and click OK. The Java Status Page opens for the selected server.

The screenshot shows the Zend Platform i5/OS Java Status Page. At the top, it displays the date and time (06 Feb 2006 13:53:10), user (Logged in as Admin), and server name (Gollum). It includes links for Refresh, Help, Stop, and Restart. Below this is a table for Java Environment variables:

Field	Value
Java Version	1.5.0_05
Java Vendor	Sun Microsystems Inc.
OS Name	Linux
OS Version	2.6.11-1.14_FC3
Class Path	/usr/local/Zend/bin/javamw.jar:
Java Home	/usr/local/ZDE/jre

Below the environment table is a section for Bridge Statistics:

Field	Value
Number of connections	3
Number of requests	28

Figure: 42 - Java Status Page

Note:

Statistical information is gathered on the Java server therefore, even after restarting the web server, the statistics are maintained. This naturally does not apply to restarting the Java server, which will restart the statistics collection from zero.

Using the active buttons provided on the Java Bridge user interface, you can Stop the Java Bridge, Restart the Java Bridge, or Refresh the Status Information shown on-screen for the selected server.

To stop the Java Bridge:

1. Click Stop.
Platform opens an information screen that tells you that the Java Bridge was successfully stopped.
2. Click OK to close the window (or the corner X).
Platform closes the window and returns to the Java Bridge main screen.

Configuring the Java Bridge

Zend Platform's Java module has two configuration parameters:

- **zend.javamw.threads** - How many worker threads the server is using; allowing this number of concurrent requests to be executed. Default is 20.
- **zend.javamw.port** - TCP port on which the server is listening. Default is 10001.

Example Script

The following example is the shell script for running JavaMW (This script should be customized when necessary):

```
export CLASSPATH=$CLASSPATH:`pwd`/javamw.jar  
java -Dzend.javamw.threads=20 -Dzend.javamw.port=10001 com.zend.javamw.JavaServer  
Add other entries into CLASSPATH if you use non-standard Java packages.
```

PHP Configuration

The PHP module uses the following configuration directives:

- **java.server_port** - TCP port on which the server is listening. Default is 10001.

Note:

This must be the same as zend.javamw.port for the server.

- **Java.ints_are_longs** - converts PHP's integer to Java's java.lang.Long. By default and if this option is off, the PHP's integers are converted to java.lang.Integer.

Usability Issues

The Java Bridge's PHP 4 module has a number of usability issues compared to native Java code, which stem from the limitations imposed by the PHP 4 language. Most of these limitations do not exist in PHP 5 and those that do will be mentioned as such.

Chain Functions Call

In pure Java, you can program the following chain function:

```
result = object.Method1().Method2().Method3();
```

In this example, the result of one method becomes the object for another method. PHP 4's Java module, however, does not allow you to write a chain function in a similar way, as per example:

```
$result = $java_object->Method1()->Method2()->Method3();
```

This is due to the fact that PHP 4 disallows chaining method calls. Instead, a chain function must be expressed as follows:

```
$result1 = $java_object->Method1();
$result2 = $result1->Method2();
$result = $result2->Method3();
```

Note:

In PHP 5, you can use chaining.

Exceptions

Since PHP 4 has no concept of exception, you may not include Java exceptions in your PHP code. However, you can use functions to deal with exceptions.

```
java_exception_get:  
http://www.zend.com/manual/function.java-last-exception-get.php
java_last_exception_clear:  
http://www.zend.com/manual/function.java-last-exception-clear.php
```

PHP 5 has a concept of exceptions and therefore can handle Java exceptions and translate them into PHP exceptions.

The following examples display the different exception scenarios and what they look like:

How Exceptions Work:

In this example exceptions are inherited from an exception class.

```
<?
try {
    $stack=new Java("java.util.Stack");
    $stack->push(1);
    $result = $stack->pop();
    print "$result\n";
    $result=$stack->pop();
```

```

} catch(Exception $ex) {
    print "Exception in pop: ";
    print $ex->getCause()->toString();
    print "\n";
}
?>

```

Caught Exceptions:

This example shows what an exception looks like when a Java Code exception is caught. This is an example of a typical exception that will appear instead of the expected PHP output when specified in the code i.e using print_r (\$exception) or var_dump (\$exception).

```

JavaException Object
(
    [message:protected] => Java Exception java.util.EmptyStackException:
java.util.EmptyStackException
    at java.util.Stack.peek(Stack.java:79)
    at java.util.Stack.pop(Stack.java:61)

    [string:private] =>
    [code:protected] => 0
    [file:protected] => /vector.php
    [line:protected] => 7
    [trace:private] => Array
        (
            [0] => Array
                (
                    [file] => /vector.php
                    [line] => 7
                    [function] => pop
                    [class] => java.util.Stack
                    [type] => ->
                    [args] => Array
                        (
                            )
                )
        )
    [javaException] => java.util.EmptyStackException Object
)

```

Uncaught Exceptions:

Uncaught exceptions are also reported but because they lack an immediate relation to a specific exception class they are less detailed and can only indicate basic details regarding the occurrence of an exception. This type of exception typically appears in your error log or wherever you have defined your php.ini to store errors.

Note:

Please refer to the PHP manual for more information regarding the php.ini error reporting definitions.

```
Fatal error: Uncaught exception 'JavaException' with message 'Java
Exception java.util.EmptyStackException:
java.util.EmptyStackException
    at java.util.Stack.peek(Stack.java:79)
    at java.util.Stack.pop(Stack.java:61)
' in /vector.php:7
Stack trace:
#0 /vector.php(7): java.util.Stack->pop()
#1 {main}
thrown in /vector.php on line 7
```

Java Array/Hashtable Objects

In PHP, arrays and hashtables are used interchangeably. This is because in PHP hashtables are indexed by integers or strings—not by objects. In Java, the key and value must be objects to be associated, so primitive types have to be converted to objects first, before parsing.

In Zend Platform's Java interface, if a method returns array/hashtable, it is immediately translated into a PHP native array/hashtable type. This means that if you want to work with a Java array/hash from PHP you cannot preserve it as a Java object. Of course, the contents are preserved, but the object identity is lost. In such a case, when an array/hash is returned, you will lose the ability to use Java methods since the array/hash loses the object identity and becomes a regular PHP array.

There are several ways to handle Java arrays and hashtable descendants. The following example shows a possible scenario of how Java arrays and hashtable descendants can be converted into PHP arrays by splitting the class pattern method and returning an array of strings which is then converted into a PHP array as follows:

```
<?
$exp = new Java("java.util.regex.Pattern");
$p = $exp->compile(":+"); // Create new pattern object
$arr = $p->split("a::b:c:::d:e"); // Use pattern to split string into array
print_r($arr);
?>
```

To deal with Array/Hashtable objects originating in Java:

Implement the code dealing with the array in Java and then call it from PHP, or encapsulate the object in a different class.

Iterators

Iterators are not handled by the Java Bridge in any special way and they are treated like any other Java object.

Common Tasks

This section describes some of the common uses for the Zend Platform Java Bridge. The usage scenarios and examples discussed here provide a framework for the Java Bridge's uses, rather than a complete picture. Real world experience indicates that companies are finding more and more applications for the Java Bridge, beyond what was initially anticipated.

Usage Scenarios

There are two usage scenarios that describe the most common applications for Zend Platform's PHP/Java Bridge:

- **Integration with Existing Java Infrastructure** — PHP is a fully featured scripting language engineered to cover virtually all of an enterprise's requirements. At the same time, many enterprises have a long history of application development in Java. Platform's Java Bridge enables enterprises to keep on using their Java infrastructure —applications, databases, business logic, and various Java servers (WebLogic, JBoss, Oracle Application Server, etc.)
- **Accessing Java Language and Architecture** — Some enterprises require the full set of PHP's capabilities, yet have a specific need for select Java based applications. SIP signaling in the communications industry or JDBC for creating connectivity to SQL databases are two examples of impressive, industry specific products. Platform's Java Bridge enables enterprises to adopt a PHP standard and to use their preferred Java based applications.

Activities

This section describes two sample activities that indicate some of what you can do with Platform's PHP/Java Bridge. In the sample activities, it is important to differentiate between Java and J2EE. The difference will impact on architecture, and in turn, on the script code.

The important differences are:

- Java is a programming language. Java applications created in Java for the enterprise are not bound to a specific framework. Therefore, it is possible and perhaps preferable for an enterprise to relocate code libraries to a Zend Platform node.
- J2EE is a structured framework for application scripts developed for J2EE. It is preferable that J2EE servers be left intact. (See the Zend Platform System Diagram above.)

Example 1: Typical Code

The code sample below is a functional example—you can run it! The example demonstrates the interaction between the PHP application and Java objects that occurs in the Java Bridge implementation.

```
<?
// create Java object
$formatter = new Java("java.text.SimpleDateFormat",
                      "EEEE, MMMM dd, yyyy 'at' h:mm:ss a zzzz");
// Print date through the object
print $formatter->format(new Java("java.util.Date"))."\n";
// You can also access Java system classes
$system = new Java("java.lang.System");
print $system."\n"; // will use toString in PHP5
```

```
print "Java version=".$system->getProperty("java.version")." <br>\n";
print "Java vendor=".$system->getProperty("java.vendor")." <p>\n\n";
print "OS=". $system->getProperty("os.name")." ".
      $system->getProperty("os.version")." on ".
      $system->getProperty("os.arch")." <br>\n";
?>
```

The example code can be understood as follows:

1. The code example is written in PHP and forms part of a PHP Web application.
2. The PHP code creates the Java object—"java.lang.System"—which is the PHP proxy.
3. The purpose of the PHP code is to print the date and system information; however, it does so through the Java object.

Example 2: A Case Study Java Bridge Performance

The Forever Times newspaper maintains a PHP-based website—let's call it [ForeverOnline.com](#). The newspaper has been searching for a real-time Stock Ticker application to add to their already successful and heavily visited website. The Forever Times Newspaper feels that real-time financial information is the one thing their web site is lacking.

Forever Times believes they have found exactly the Stock Ticker application they need. The application provides up-to-date quotations from all the major markets, currency rates, and even links to some of the local exchanges. However, the application is written in Java and uses existing Java libraries.

Forever Times realizes that a PHP based Web implementation that handles Java requests—a Java Bridge—is their best bet. At the same time, they are concerned that the performance of their Website remains optimal. To Forever Times' horror, in testing the new application, they found that loading the site with user-requests for the stock ticker slows down the performance of the whole Website.

The following code example illustrates how Platform's Java Bridge applies to this business scenario and others like it:

```
<?
// create Java object
$stock = new Java("com.ticker.JavaStock");
// call the object
$news = $stock->get_news($_GET['ticker']);
// display results
foreach($news as $news_item) {
print "$news_item<br>\n";
}
?>
```

The example code can be understood as follows:

- The code example is written in PHP and forms part of a PHP Web application.
- The PHP code creates the Java object—"com.ticker.JavaStock"—which is the PHP proxy.
- Requests come into the PHP based Website – [ForeverOnline.com](#) – which then references

the Stock Ticker application.

- Stock Ticker references a custom object—`get_news`—in the JVM library. This is all in native Java.
- The PHP code then outputs the results on the Website.

The Typical Java Bridge Implementation and the Zend Platform's Java Bridge Implementation diagrams below show how Forever Times' concern about performance is addressed, through the Zend Platform Java Bridge architecture. The diagrams focus on how problems in scalability arise in a typical Java Bridge Implementation.

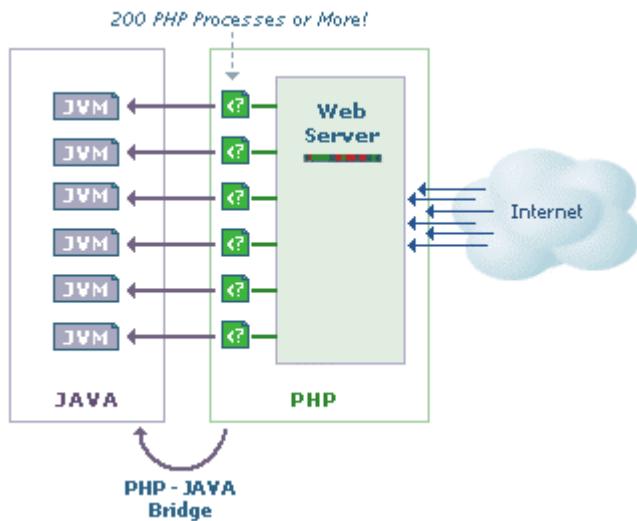


Figure: 43 - Typical Java Bridge Implementation

Zend Platform's Java Bridge Implementation diagram shows how scalability issues are addressed in the Zend Platform Java Bridge.

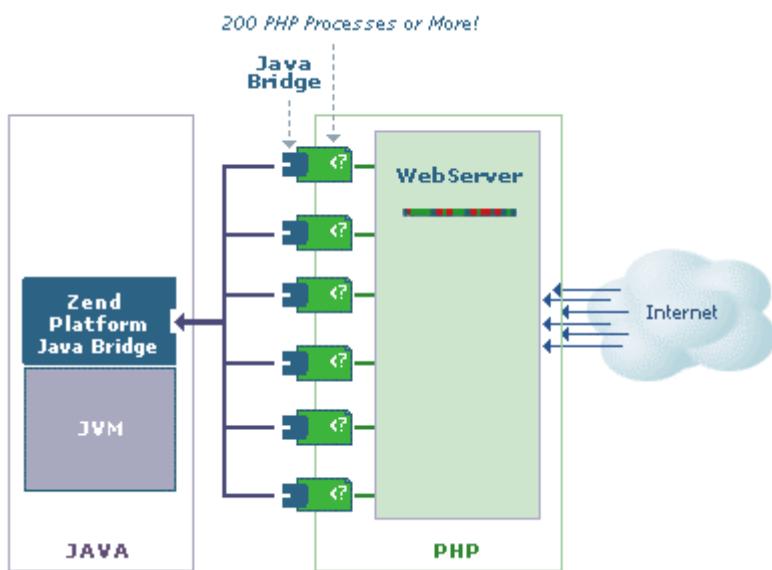


Figure: 44 - Zend Platform's Java Bridge Implementation

Note:

While the single JVM constitutes a single point of failure, the fact is Zend's PHP-Java connection is the most robust on the market. Failures in systems of this type generally tend to occur when the Java Server is overloaded, rather than as a result of glitches in the applications. Zend Platform's system architecture insures performance by diminishing overhead. However, in the event of failure, the Java Bridge GUI supports a Restart feature that makes monitoring the status of the Java Server and restarting quick and simple. One last point: if the failure was caused by a glitch in the application, the same thing would most likely occur in each of the JVMs in the non-Zend system!

Example 3: Case Study in Management Integration

A company called FlowerPwr.com, sells flowers over the Internet. They are a successful East coast based firm that has an aggressive management profile. They are currently in the process of acquiring a West coast competitor—let's call it Yourflowers.com—that provides a similar service.

FlowerPwr.com has its own website, and its various enterprise applications were written in PHP. Yourflowers.com has its own Website, however all its applications are Java based and were developed for J2EE. They have their own J2EE application server. FlowerPwr.com needs to begin operating as an integrated commercial entity as soon as possible in a way that conceals the fact that the companies have merged.

Platform's Java Bridge offers a solution. Using Zend Platform, FlowerPwr.com can create a common portal in PHP. The company can leave Java up and running and take full advantage of their acquisition's existing Java services. FlowerPwr.com can do this over an existing portal using PHP.

The following code example illustrates how Platform's Java Bridge can apply to this business scenario and others like it:

```
<?

// EJB configuration for JBoss. Other servers may need other settings.

// Note that CLASSPATH should contain these classes

$envt = array(
    "java.naming.factory.initial" => "org.jnp.interfaces.NamingContextFactory",
    "java.naming.factory.url.pkgs" => "org.jboss.naming:org.jnp.interfaces",
    "java.naming.provider.url" => " jnp://yourflowers.com:1099");
$cxt = new Java("javax.naming.InitialContext", $envt);

// Try to find the object
$obj = $cxt->lookup("YourflowersBean");

// here we find an object - no error handling in this example
$rmi = new Java("javax.rmi.PortableRemoteObject");
$home = $rmi->narrow($obj, new Java("com.yourflowers.StoreHome"));
// $hw is our bean object
$store = $home->create();
// add an order to the bean
$store->place_order($_GET['client_id'], $_GET['item_id']);
print "Order placed.<br>Current shopping cart: <br>";
```

```
// get shopping cart data from the bean  
$cart = $store->get_cart($_GET['client_id']);  
foreach($cart as $item) {  
print "$item[ 'name' ]: $item[ 'count' ] at $item[ 'price' ]<br>\n";  
}  
// release the object  
$store->remove();  
?>
```

The example code can be understood as follows:

1. The code example is written in PHP and forms part of a PHP Web application.
2. The PHP application first initializes an operation with the EJB, located at a specific URL that has the name: "jnp://yourflowers.com:1099."
3. The code then specifies the bean—YourflowersBean—that the application will look for.
4. Next, the bean object is returned from the EJB server.
5. The application then calls methods—in this case, the Java application includes two functions:
 - *place_order receiving two numbers* — client ID and the item ID to add to shopping cart
 - *get_cart receiving one number* — client ID and returning the list of the items placed in the shopping cart so far.

After script execution the referenced class may be disposed.

This completes the Java Bridge chapter of the User Guide. In this section we have described how provide and configure Zend Platform in order to obtain interoperability with other existing legacy or backend applications written in Java.

Chapter 6 - APIs and Directives

IN THIS CHAPTER...

ZEND PLATFORM APIs:

- ACCELERATOR FUNCTIONS
- OUTPUT CACHE FUNCTIONS
- MONITOR FUNCTIONS
- ZDS (ZEND DOWNLOAD SERVER)

ZEND PLATFORM DIRECTIVES:

- ACCELERATOR DIRECTIVES
- MONITOR DIRECTIVES
- GUI DIRECTIVES
- COLLECTOR CENTER DIRECTIVES
- DEBUGGER DIRECTIVES
- ZDS DIRECTIVES

This chapter is a reference chapter for Zend Platform APIs and directives.

Zend Platform APIs

Accelerator Functions

`accelerator_set_status`

`void accelerator_set_status(bool status)`

- Description: Disable/enable the Code Acceleration functionality at run time.
- Return Values: none
- Parameters: status - if false, Acceleration is disabled, if true - enabled

Output Cache Functions

`output_cache_disable()`

- Description: Disables output caching for currently running scripts.
- Return Values: none
- Parameters: none

`output_cache_disable_compression()`

- Description: Does not allow the cache to perform compression on the output of the current page. This output will not be compressed, even if the global settings would normally allow compression on files of this type.
- Return Values: none
- Parameters: none

`output_cache_fetch()`

`string output_cache_fetch(string key, string function, int lifetime)`

- Description: Gets the code's return value from the cache if it is there, if not - run function and cache the value.

- Return Values: function's return
- Parameters: key - cache key, function - PHP expression, lifetime - data lifetime in cache (seconds)

output_cache_output()

string output_cache_output(string key, string function, int lifetime)

- Description: If they cache for the key exists, output it, otherwise capture expression output, cache and pass it out.
- Return Values: expression output
- Parameters: key - cache key, function - PHP expression, lifetime - data lifetime in cache (seconds)

output_cache_remove

bool output_cache_remove(string filename)

- Description: Removes all the cache data for the given filename.
- Return Values: true if OK, false if something went wrong
- Parameters: filename - full script path on local file system

output_cache_remove_url

bool output_cache_remove_url(string url)

- Description: Remove cache data for the script with given URL (all dependent data is removed)
- Return Values: true if OK
- Parameters: url - the local url for the script

output_cache_remove_key

bool output_cache_remove_key(string key)

- Description: Remove item from PHP API cache by key
- Return Values: true if OK
- Parameters: key - cache key as given to output_cache_get/output_cache_put

output_cache_put

bool output_cache_put(string key, mixed data)

- Description: Puts data in cache according to the assigned key.
- Return Values: true if OK
- Parameters: key - cache key, data - cached data (must not contain objects or resources)

output_cache_get

mixed output_cache_get(string key, int lifetime)

- Description: Gets cached data according to the assigned key.
- Return Values: cached data if cache exists, false otherwise
- Parameters: key - cache key, lifetime - cache validity time (seconds)

output_cache_exists

bool output_cache_exists(string key, int lifetime)

- Description: If data for assigned key exists, this function outputs it and returns a value of true. If not, it starts capturing the output. To be used in pair with output_cache_stop.

- Return Values: true if cached data exists
- Parameters: key - cache key, lifetime - cache data validity time (seconds)

output_cache_stop

- Description: If output was captured by output_cache_exists, this function stops the output capture and stores the data under the key that was given to output_cache_exists().

Monitor Functions

monitor_pass_error

void monitor_pass_error(integer \$errno, string \$errstr, string \$errfile, integer \$errline)

- Description: Should be called from a custom error handler to pass the error to the monitor. The user function needs to accept two parameters: the error code, and a string describing the error. Then there are two optional parameters that may be supplied: the filename in which the error occurred and the line number in which the error occurred.

monitor_set_aggregation_hint

void monitor_set_aggregation_hint(string \$hint)

- Description: Limited in the database to 255 chars, this API is a global variable that can be set anywhere and in any hierarchy. The purpose of this API is to incorporate locations of occurrences in the script. This API is used when there are events that require the location in the script for diagnosing the reason behind the event occurring. For example: Global Events require the application that generated the event. Adding the Hint API can assist in the identification process. This string that is supplied by the user to differentiate between pages that have the same URL but different parameters.
- Return Values: If the user did not supply a hint the default hint is an empty string.
- Parameters: \$hint

monitor_custom_event

void monitor_custom_event(string \$class, string \$text[, integer \$severe, mixed \$user_data])

- Description: Custom Events are used to generate an event whenever the API function monitor_custom_event() is called from the PHP script. This event type enables the generation of an event on occurrences that are not necessarily built-in Zend Platform events (error and performance issues). Custom Events are used whenever you decide that it is significant to generate an event in a certain situation. Each event type is given a name for easy identification (\$type).
- Parameters: \$class – helps to define several types of custom events. This description will be showed in the PHP Intelligence, Event List and in the Event Details (report). \$text - error text used to describe the reason for the event. This text will appear in the Event Details. \$severe - the severity level of the triggered event, default value is Severe. \$user_data - adds a PHP variable that will be viewed in the Event Details screen (in Event Context-> Variables->User Defined). This forms the stored Event Context (similar to the information obtained in a PHP error event).

register_event_handler

*boolean register_event_handler(\$event_handler_func
[\$handler_register_name],\$event_type_mask)*

- Description: Allow you to register a user function as an event handler. When a monitor event is triggered all the user event handler are called and the return value from the

handler is saved in an array keyed by the name the event handler was registered under. The event handlers results array is saved in the event_extra_data table.

- Return Value: TRUE on success and FALSE if an error occurs.
- Parameters: The first argument is a callback function that will be call when the event is triggered, object methods may also be invoked statically using this function by passing array (\$objectname, \$methodname) to the function parameter. The second (optional) argument is name this function is registered under - if none is supplied, the function will be registered under it's own name. The third (optional) parameter is a mask of event types that the handler should be called on by default it's set to MONITOR_EVENT_ALL.

unregister_event_handler

boolean unregister_event_handler(string handler_name)

- Description: Allow you to un-register an event handler.
- Return Value: TRUE on success and FALSE if no handler we registered under the given name.
- Parameters: string handler_name - the name you registered with the handler you now wish to un-register.

ZDS (Zend Download Server)

zend_send_file

bool zend_send_file(string filename[, string mime_type])

- Description: Send a file using ZDS
- Return Value: FALSE if sending file failed, does not return otherwise
- Parameters: filename - path to the file, mime_type - MIME type of the file, if omitted, taken from configured MIME types file

Note:

The ZDS is not currently supported in Windows.

Zend Platform Directives

Accelerator Directives

"zend_accelerator.max_wasted_percentage"

Description: Max percentage of "wasted" memory until restart is scheduled

"zend_accelerator.max_warmup_hits"

Description: How many hits are considered 'warmup' (for statistics)

"zend_accelerator.consistency_checks"

Description: Check cache's checksum each N requests

"zend_accelerator.force_restart_timeout"

Description: Time to wait for cache being unused when restart is scheduled (seconds)

"zend_accelerator.perform_timings"

Description: Collect performance statistics

"zend_accelerator.validate_timestamps"

Description: Check file timestamps

"zend_accelerator.max_cached_filesize"

Description: Max cached size for content cache (Kbytes)

"zend_accelerator.revalidate_freq"

Description: How often to check file timestamps on Windows (seconds)

"zend_accelerator.min_free_disk"

Description: Min disk space to leave free for content cache (in M or %)

"zend_accelerator.php_extensions"

Description: List of extensions to consider for content cache when directory is configured

"zend_accelerator.user_blacklist_filename"

Description: Path for a file that contains a list of files not to accelerate

"zend_accelerator.compress_blacklist_filename"

Description: Path for a file that contains a list of files not to compress

"zend_accelerator.compression"

Description: Enable compression for content cached files

"zend_accelerator.compress_all"

Description: Enable compression for accelerated files

"zend_accelerator.enabled"

Description: Enable acceleration

"zend_accelerator.output_cache_enabled"

Description: Enable content caching

"zend_accelerator.max_accelerated_files"

Description: Maximum number of keys (scripts) in accelerator hash table

"zend_accelerator.mmap_base_file"

Description: Windows: location of map address file

"zend_accelerator.httpd_uid"

Description: UID of the httpd process

"zend_accelerator.memory_consumption"

Description: Accelerator shared memory block size (Mbytes)

"zend_accelerator.allow_noshm"

Description: Allow running in "no shared memory mode" (CGI, CLI)

"zend_accelerator.output_cache_config"

Description: Content cache configuration file

"zend_accelerator.output_cache_dir"

Description: Content cache storage directory

"zend_accelerator.use_cwd"

Description: Use current directory as a part of script key

"zend_accelerator.preferred_memory_model"

Description: Shared memory model to use

"zend_accelerator.dups_fix"

Description: Use hack to prevent "duplicate definition" errors

"zend_accelerator.cgi_base_shm_address"

Description: The base address for the CGI/CLI shared memory block

Monitor Directives

"zend_monitor.max_var_len"

Description: Maximum variable length for collected data in POST/SERVERS. Limit applies to each single value.

"zend_monitor.warmup_requests"

Description: Number of requests until monitor would use averaging statistics to produce events

"zend_monitor.load_sample_freq"

Description: Frequency of checking for load events (seconds)

"zend_monitor.rotate_freq"

Description: Frequency for rotating monitor internal log files (seconds)

"zend_monitor.reconnect_timeout"

Description: How long monitor will wait until trying to restore broken connection to central (seconds)

"zend_monitor.watch_functions"

Description: List of functions to watch for time events (@file reads list from file)

"zend_monitor.watch_results"

Description: List of functions to watch for failure return events (@file reads list from file)

"zend_monitor.collector_host"

Description: Hostname for central

"zend_monitor.collector_port"

Description: Port for central

"zend_monitor.log_dir"

Description: Directory where monitor logs will be kept

"zend_monitor.server_key"

Description: Filename for local SSL key

"zend_monitor.server_cert"

Description: Filename for local SSL certificate

"zend_monitor.collector_cert"

Description: Filename for central SSL certificate

"zend_monitor.enable"

Description: Monitoring is enabled

"zend_monitor.error_level"

Description: Errors reported as events

"zend_monitor.error_level.severe"

Description: Errors reported as severe events

"zend_monitor.silence_level"

Description: If 1, does not report errors when error reporting is 0. If 2, doesn't report errors only if @ is used.

"zend_monitor.max_script_runtime_load_cutoff"

Description: Load value which would suppress time-related events

"zend_monitor.report_variables_data"

Description: Which variables to report (*)

"zend_monitor.max_script_runtime"

Description: Script runtime above which event is produced (ms)

"zend_monitor.max_function_runtime"

Description: Function runtime above which event is produced (ms)

"zend_monitor.max_memory_usage"

Description: Memory usage above which event is produced (K)

"zend_monitor.max_load"

Description: Load above which event is produced

"zend_monitor.max_script_runtime.severe"

Description: Script runtime above which severe event is produced (ms)

"zend_monitor.max_function_runtime.severe"

Description: Function runtime above which severe event is produced (ms)

"zend_monitor.max_memory_usage.severe"

Description: Memory usage above which severe event is produced (K)

"zend_monitor.max_load.severe"

Description: Load above which severe event is produced

"zend_monitor.max_time_dev"

Description: Deviation from average script runtime above which event is produced (%)

"zend_monitor.max_output_dev"

Description: Deviation from average output size above which event is produced (%)

"zend_monitor.max_mem_dev"

Description: Deviation from average memory usage above which event is produced (%)

"zend_monitor.max_time_dev.severe"

Description: Deviation from average script time above which severe event is produced (%)

"zend_monitor.max_output_dev.severe"

Description: Deviation from average output size above which severe event is produced (%)

"zend_monitor.max_mem_dev.severe"

Description: Deviation from average memory usage above which severe event is produced (%)

"zend_monitor.mem_threshold"

Description: If memory usage below this value, no deviation events are produced

"zend_monitor.time_threshold"

Description: If script runtime below this value, no deviation events are produced

"zend_monitor.output_threshold"

Description: If output size below this value, no deviation events are produced

"zend_monitor.event_overload_threshold"

Description: If more than 1000 events happen in this time (seconds), extra events will be dropped

"zend_monitor.disable_script_runtime_after_function_runtime"

Description: Disable "script slow" event after "function slow" event happened

"zend_monitor.tmp_dir"

Description: Directory where monitor temp files are written

(*) G - GET, P - POST, C - COOKIE, R - RAW_POST_DATA, E - ENV, V - SERVER, S - SESSION, F - FILES

Zend Monitor Event Types

For each event type there is a zend_monitor.<event_type>.

zend_monitor.<event_type> - can be set to off then this event type will not be reported.
E.g.: zend_monitor.memsize.enable = Off however the same settings can be easily defined from PHP Intelligence | Event Triggers.

The following list displays the event types and the respective directive for enabling and disabling Events:

- **Slow Script Execution Absolute** - zend_monitor.longscript.enable
- **Slow Script Execution Relative** - zend_monitor.devscript.enable
- **PHP Error** - zend_monitor.zenderror.enable
- **Function/Database Error** - zend_monitor.funcerror.enable
- Slow Function Execution/Slow Query Execution - zend_monitor.longfunction.enable
- **Excess Memory Usage (Absolute and Relative)** -
zend_monitor.devmem.enablezend_monitor.memsize.enable
- **Inconsistent Output Size** - zend_monitor.outsize.enable
- **Load Average** - zend_monitor.load.enable
- **Custom Event** - zend_monitor.custom.enable

Note:

All event types are enabled, by default. zend_monitor.enable when turned off will disable all event reporting activity.

GUI Directives

zps.install_dir

Description: The place the Zend directory was installed to (Platform/ZPS)

studio.install_dir

Description: The place the Zend directory was installed to (Studio Server)

zend_gui.language

Description: Language code the GUI uses for texts (i.e. en for English)

zend_gui.language_charset

Description: If set, all the GUI files will send a Content-Type header with this charset also used to send specific charset in Email (should be used in the Japanese version)

zend_gui.ini_modifier

Description: The path where the ini_modifier util is zend_central.error_logging

Description: If enabled, the Zend Platform GUI will log errors into the file'zend_central_error_log' that is located in the <install-dir>/logs directory.

zend_central.gui_address

Description: The full address of the central GUI, this address is used by the node to access the central GUI (login process) The address is in http(s)://host:port/path format

zend_central.node_address Each node has this directive set with his address, the same address that he gave the central during installation (this is the way the central identify the server in the DB) The address is ONLY the hostname/IP address

zds.your_servers_max_clients

Description: Use for the ZDS tests in the GUI (Performance section in Platform), to "know" what is the value of the maxClients of the server

Collector Center Directives

zend_monitor.collector_cert

Description: Certificate file for the CC

zend_monitor.collector_key

Description: Private key file for the CC

zend_monitor.collector_port

Description: Port to listen

zend_monitor.server_key_dir

Description: Dir to store node keys

zend_monitor.events_db

Description: Event DB URI

zend_monitor.pull_freq

Description: How often to pull node data (seconds)

zend_monitor.ping_freq

Description: How often to check node availability (seconds)

zend_monitor.log_dir

Description: Dir to store logs

zend_monitor.gui_dir

Description: Dir where GUI files reside

zend_monitor.event_lifetime

Description: How long until event is considered too old (seconds)

zend_monitor.cleanup_freq How often to clean up old events (event count)

Debugger Directives

"zend_debugger.allow_hosts"

Description: Hosts allowed to connect (hostmask list)

"zend_debugger.deny_hosts"

Description: Hosts denied to connect (hostmask list)

"zend_debugger.allow_tunnel"

Description: Hosts allowed to use tunnel process (hostmask list)

"zend_debugger.expose_remotely"

Description: Which client can know debugger is installed

"zend_debugger.max_msg_size"

Description: Maximum message size accepted by Debugger

"zend_debugger.httpd_uid"

Description: UID for the httpd process

ZDS Directives

"zds.enable" Enable

Description: ZDS file serving

"zds.mime_types_file"

Description: Location of the MIME types file

"zds.log_file"

Description: Log file

"zds.min_file_size"

Description: Minimal file size to serve via ZDS process (smaller files served via Apache)

"zds.disable_byterange"

Description: Disable handling byte-range requests (all requests would return entire file)

"zds.mmap_chunk" Memory chunk to map when serving file, in K. Bigger chunks imply higher memory usage by ZDS.

"zds.nice"

Description: Priority of ZDS server process. Higher number means lower priority.

"zds.child_max"

Description: Maximum number of ZDS sub-processes

"zds.poll_delay"

Description: Delay between poll invocations, in order to enable other processes to run better

"zds.uid" UID of the ZDS file server

Chapter 7 - Tutorials

IN THIS CHAPTER...

INTEGRATING EXISTING AND LEGACY APPLICATIONS

CALLING AN EJB ON WEBSPHERE FROM PHP

PARTIAL AND PREEMPTIVE PAGE CACHING

This section of the User Guide is dedicated to tutorials on different subjects.

Tutorial Feedback

Please send us your opinion and suggestions for new tutorials by e-mail to:
documentation@zend.com.

Integrating Existing and Legacy Applications

This tutorial details the integration of Zend Platform's Event Details screens with other legacy applications.

Reproducing and resolving bugs, one of the most problematic challenges of development, is often time consuming, and in most cases, almost impossible when information is not collected at the time of the occurrence. PHP Intelligence is an event driven system that provides real-time analysis of PHP applications. By enabling you to obtain immediate insight into your PHP applications, PHP Intelligence provides a fast and efficient means to reproduce and resolve problems, while maintaining a complete audit trail of the occurrence's details.

PHP Intelligence proactively alerts you to problematic occurrences in your application. This means that if you are a Developer or System Administrator you will not need to monitor the Zend Platform console at all times—instead, the information comes to you! An event, containing the audit trail of an occurrence, can be made known to you through an Email Notification. If you require full event details available outside of the Zend Platform console, an Event Details screen can be published to a URL in XML format. Both Event Details screens contain aggregated information relevant to the occurrence of an event, or in other words "Full Problem Context": Event type, Event ID, Timestamp, Severity, number of occurrences, etc.

Full Problem Context provides valuable information for the entire PHP application lifecycle (development, production and deployment). Exposing the source of an occurrence along with the ability to drill-down and investigate details pertaining to an event's location, time and context, provides in-depth insight to the reasons why the event occurred and a basis for resolving the issue.

PHP Intelligence includes the following Event Details screen Types: Slow Script Execution, PHP Errors, Function Errors, Memory Usage, Database Errors, Query Execution, Output Sizes, Load Averages and more...

Each Event Details screen Type includes basic and event-specific details such as: event type, event ID, Timestamp, Severity, number of occurrences, error type, error text, triggered value, load average, Source File Line, Script Name, Host URI, Vardata Type & Name, Function Name, Argument Numeric Value, Function, Included Files, Backtrace, etc.

Contents of the XML output can be easily utilized and integrated to provide an information feed to various legacy systems such as: Bug tracking systems for development and QA, CRM applications for managing customer care, management systems such as Tivoli and HP OpenView for system health information, and most commonly, generic monitoring systems such as Nagios or BigBrother that only provide OS service information.

Using event information, developers and administrator teams have a single point of reference to streamline the maintenance workflow. You can further enhance your development lifecycle by debugging your PHP code referenced in Event Details screens directly through the built-in integration with Zend Studio. This feature includes debug capabilities that enable you to add watches, define conditional breakpoints, view the stack trace and step into the source code to immediately debug the problem.

Zend Platform's XML output enables information to be easily interchanged. Using "Event Details", developers can be sure that information pertaining to code, database and performance issues can be easily reused in a multitude of applications. Examples of this use include sending SMS messages containing event details, or triggering a mailing system to send a promotional gift to a customer who encountered a performance problem. Done by, extracting customer ID information provided to you in the Event Details screen (cookies).

Event Details screens are delivered as XML, by defining the relevant action ("Submit Report to the Specified URL") for an event. The report information is submitted as XML data to the specified URL. Submission is done using the POST method, and the data is supplied through a variable named 'event_data'. This variable is accessible in PHP through `$_POST['event_data']`.

XML reports are structured as follows:

Each attribute is included if it exists in the Event Details screen:

```
<?xml version="1.0" ?>
<event type event_id class timestamp time severity>
```

If there is an error:

```
<error type>error text</error>
<stats triggered_value avg load_average/>
```

If there is a source file:

```
<source file line/>
<script name host uri>
    <vardata type name value/>
</script>
```

If there is a function:

```
<function name>
    <args>
        <arg num value/>
    </args>
</function>
```

If there are included files:

```
<included_files>
    <file name\>
</included_files>
```

If there is a backtrace for this event:

```
<backtrace>
    <call depth function file line/>
</backtrace>
</event>
```

By viewing the XML tagged file as fielded text, the fielding makes it possible to break Event Details screens down to their component parts to any degree of granularity for storage in a database. Once in the database, the data can be utilized by another application.

The following example shows how Event Details data can be extracted from an XML file and inserted into a database for use in a different system (could be any system based on a database, such as: Bug Tracking, CRM, management or any other application).

```
<?PHP
$event_xml_data = (isset($_POST['event_data']))?$_POST['event_data']:null;
if (!$event_xml_data) {      // no Event Context arrived
    die();
}
$xml = simplexml_load_string($event_xml_data);
```

Implement different behaviors according to Class

```
$event_type = (string) $xml['type'];
// if this event is a Custom Event, we may implement different behaviors according to
the Custom Event's class
if ($event_type == 'custom') {
$custom_class = (string) $xml['class'];
switch ($custom_class) {
// different behaviors according to the class
}
}
// get the new event id
$event_id = (int) $xml['event_id'];
// insert a new event with its genreal info (type and timestamp) to the db
insert_new_event_into_db($event_id,$event_type ,(int)
$xml['timestamp']);
// parse the function parameters of the function where the event occured
$function_parameters = array();
foreach ($xml->function->args->arg as $arg) {
    $function_parameters[(int) $arg['num']] = (string) $arg['value'];
}
// insert the function data (function name and parameters, where the
event occured) to the db
update_event_function_data($event_id, (string) $xml->function['name'],
$function_parameters);
```

```
/**  
 * insert a new event (with some genreal info) to the db  
 *  
 * @param int $id id of the new event in the ZendPlatform events database  
 * @param string $type the event type  
 * @param int $timestamp the unix timestamp when the event occured  
 */  
function insert_new_event_into_db($id,$type,$timestamp) {  
}  
/**  
 * update a specific event function data in the database  
 *  
 * @param int $id the event id we want to update  
 * @param string $function_name name of the function where the event occured  
 * @param array $function_params array of the function parameters (num  
=> value)  
 */  
function update_event_function_data($id,$function_name,$function_params) {  
}  
?>
```

The first part of the example uses a PHP 5 Simple XML extension to parse XML to PHP objects that can be processed with normal property selectors and array iterators. The second part extracted data from the event XML data, and inserted it into a database.

As this tutorial demonstrates, XML Event Details generated by the PHP Intelligence component of Zend Platform, provides Developers and System Administrations a single point of reference for production environments, and to streamline maintenance workflow. In environments where multiple management applications are an everyday reality, Zend Platform provides a flexible information feed to legacy systems, relieving the overhead normally required to integrate with these applications.

Calling an EJB on Websphere from PHP

This tutorial reviews the steps needed to call an EJB on WebSphere from PHP using the Java Bridge. These instructions assume that the developer has a system(s) with WebSphere, PHP, and the Zend Platform installed.

To call an EJB on Websphere from PHP, a script file needs to be created. This script should start the javaMW server with the correct settings to run a Websphere client.

Some of the important things to remember are:

1. Use IBM's java that ships with the Application Client to run the javaMW server.
2. The jars containing the client classes for any EJB that is to be called need to be classpath of the javaMW server.
3. The jars and environment variables for the WebSphere application client runtime need to be on the command line staring the javaMW server.

The following is an example script file for starting the javaMW server with the WebSphere runtime configuration options and the jars needed to call the Basic Calculator Technology sample shipped with Websphere.

Note:

This script is a modified version of the Basic Calculator Thin Client script file shipped with the Websphere Client Install.

```
#!/bin/sh
. /opt/IBM/WebSphere/AppClient/bin/setupClient.sh
# Change the PROVIDER_URL to point to this machine or another server.
if [ "${SERVERPORTNUMBER}" != "" ]
then
    PROVIDER_URL=iiop://$DEFAULTSERVERNAME:$SERVERPORTNUMBER
else
    PROVIDER_URL=iiop://$DEFAULTSERVERNAME
fi
"${JAVA_HOME}/bin/java" $WAS_LOGGING -classpath
/usr/local/Zend/bin/javamw.jar:/opt/IBM/WebSphere/AppClient/samples/lib/Tec
hnologySamplesThinClient/BasicCalculatorClientCommon.jar:/opt/IBM/WebSphere/
AppClient/samples/lib/TechnologySamplesThinClient/BasicCalculatorThinClient.
jar:/opt/IBM/WebSphere/AppClient/samples/lib/TechnologySamplesThinClient/Bas
icCalculatorEJB.jar
-Djava.ext.dirs="$WAS_EXT_DIRS" -Djava.naming.provider.url=$PROVIDER_URL
-Djava.naming.factory.initial=com.ibm.websphere.naming.WsnInitialContextFac
tory
-Dzend.javamw.threads=20 -Dzend.javamw.port=10001 "$SERVER_ROOT"
"$CLIENTSAS" com.zend.javamw.JavaServer
```

Instructions

- Start the javaMW server using the script.
- Write a PHP client, which uses the Java Bridge functionality to call the ejb running on Websphere.

The following is an example PHP client for calling the Basic Calculator Technology sample.

```
<?php
    // Get the provider URL and Initial naming factory
    // These properties were set in the script that started the Java Bridge
    $system = new Java("java.lang.System");
    $providerUrl = $system->getProperty("java.naming.provider.url");
    $namingFactory = $system->getProperty("java.naming.factory.initial");
    $envt = array(
        "javax.naming.Context.PROVIDER_URL" => $providerUrl,
        "javax.naming.Context.INITIAL_CONTEXT_FACTORY" => $namingFactory,
    );
    // Get the Initial Context
    $ctx = new Java("javax.naming.InitialContext", $envt);
    // find the EJB
    $obj = $ctx->lookup("WSSamples/BasicCalculator");
    // Get the Home for the EJB
    $rmi = new Java("javax.rmi.PortableRemoteObject");
    $home = $rmi->narrow($obj, new
Java("com.ibm.websphere.samples.technologysamples.ejb.stateless.basiccalculator.BasicCalculatorHome"));
    // Create the Object
    $calc = $home->create();
    // Call the EJB
    $num = $calc->makeSum(1,3);
    print ("<p> 1 + 3 = $num </p>");
?>
```

This tutorial detailed how developers can use EJB on WebSphere from PHP using the Java Bridge. The two steps are: to create a script file to start the JavaMW server and to write a PHP client which uses the Java Bridge functionality to call the EJB running on WebSphere.

Partial and Preemptive Page Caching

This tutorial will review one of Zend Platform's most powerful features, Partial Page Caching. Partial Page Caching is used in cases where it is impractical or impossible to cache the entire output such as when sections of the script are fully dynamic or when the conditions for caching the script are too numerous. An example of this type of usage is when some of the output is a form, that has credit card numbers, addresses and all kinds of information that for security reasons, should not be cached.

The following tutorial is a step-by-step guide to mastering Partial Page Caching.

Inside this tutorial, you will find several ways to cache your output:

- [Partial Page Caching APIs](#) – a general overview of the Caching APIs with usage examples.
- [Action Based Partial Page Caching](#) – Cache using buttons and conditions

Partial Page Caching APIs

Partial Page Caching can also be achieved using the following functions for almost all situations:

Output Caching Functions

Function	Action
<code>output_cache_fetch()</code>	Gets the code's return value from the cache, if it is there.
<code>output_cache_output()</code>	Calls a function and checks if the function exists in the cache. Yes – Print. No – Puts function output in cache and prints.
<code>output_cache_exists()</code>	Checks if the key exists in the cache. Yes – Print. No – Runs code, output in cache and prints until it reaches the stop command: <code>output_cache_stop()</code> .
<code>output_cache_stop()</code>	Indicates the end of a block of code.

Data Caching Functions

Function	Action
<code>output_cache_put()</code>	Enters a single variable into the cache
<code>output_cache_get()</code>	Gets the Variable from the cache at the end of its lifetime.

Invalidate Cache

Function	Action
output_cache_remove_key (string key)	Respectively remove items from the cache according to their type (Key, URL or File)

The partial caching functions are divided into two groups, output caching and data caching. This document will explain each of the two groups and give practical examples of each function call.

Output Caching

The first group of functions are the output caching functions. These functions capture the output from a function or block of PHP code and cache it. These functions are:

```
output_cache_output()
output_cache_exists()
output_cache_stop()
```

Output caching functions allow programmers to remove the execution of blocks of code with static output, such as looping over and printing the day's news headlines. This output changes infrequently, so instead of reprocessing it for every user caching allows PHP to skip execution and print the results.

Prototype:

```
void output_cache_output(string key, string code, int lifetime)
```

The first time *output_cache_output()* is called, it will execute the function defined in argument two, and store any output in the cache under the retrieval key specified in argument one. Each subsequent call to *output_cache_output ()* with the same value as argument one will result in the output of this cached data instead of the execution of the function in argument two, until the cache lifetime in argument three expires.

output_cache_output () is typically used to capture the output created by a function call. In order to use it, you would need to wrap a section of code as a function. When you call *output_cache_output ()*, it will call this function and cache its output.

output_cache_output () takes three arguments:

1. The key value with which this output will be cached.
2. The function call.
3. The cache lifetime in seconds.

output_cache_output() has no return value.

Usage Example

```
<?
function content($time) {
/* Create a function to Wrap the code that produces
the output */

print "<p>Cached Time: $time </p>";
/* The actual value of $time will be printed
```

```

        only once every 30 seconds. The output from
        the print statement will be cached, and the
        function call will be ignored until the cache
        lifetime expires */
}

$time = time();
/* Get current time, in seconds */
print "<p>Current Time: $time </p>";
/* Print the real current time */
output_cache_output("Current Time", "content($time);", 30);
/* Cache all output for the function content() and
   store it based on the key "Current Time" for 30
   seconds */

?>

```

Usage Notes:

output_cache_output() is used to cache the output generated from functions. To utilize it, wrap a section of code (which generates the output you wish to cache) as a function. Any output statements in this new function will be captured into a buffer and stored as cached data with the key specified.

The other two functions, *output_cache_exists()* and *output_cache_stop()*, are used in tandem to simplify the task of caching output from a given section of code.

Prototype:

```

boolean output_cache_exists(string key, int lifetime)
void output_cache_stop()

```

output_cache_exists() is called from a conditional statement. The conditional statement should wrap the section of code producing the output you are intending to cache.

output_cache_exists() takes two arguments:

1. The key value with which this output will be cached.
2. The cache lifetime in seconds.

output_cache_exists() returns a boolean. TRUE is the key exists, FALSE if it doesn't.

output_cache_stop() takes no arguments.

output_cache_stop() has no return value.

Usage Example

```

<?
if (!output_cache_exists("Some_Key2", 3)) {
    echo time();
    output_cache_stop(); // Stop buffering...
}
?>

```

Data Caching

In cases where caching the output from a script isn't possible, we offer a set of functions for caching data. These functions are `output_cache_fetch()`, `output_cache_put()` and `output_cache_get()`.

Data caching allows programmers to skip execution of repetitive database calls, increasing script performance and reducing overhead on the database. Typical uses of data caching include caching user preferences, caching product pricing, or any SQL call which changes infrequently.

Prototype:

```
string output_cache_fetch(string key, string code, int lifetime)
```

`output_cache_fetch()` works in a similar manner to the output caching function, `output_cache_output()`. The major difference is that instead of caching the output from the function it caches the return value as a string.

`output_cache_fetch()` receives 3 arguments:

1. Unique identifier string for the data (string)
2. PHP code to be cached (string)
3. Cache lifetime in seconds (integer)

`output_cache_fetch()` returns a string containing the return value of the cached code section as defined in argument 2. The ID, defined in argument 1, serves to differentiate the code section and give it a name. Lifetime, defined in argument 3, is handled in the same way the Zend Performance module handles cache lifetimes for all cached files -- cached copies older than the lifetime will be refreshed when the function is called.

Note:

Unlike normal caching only the return value of the given PHP code is cached.

Usage Example

```
<?
function get_content($time, $sec, $usec) {
    /* I define an arbitrary function which
     * returns data. */

    $data = array();
    $data["time"] = $time;
    $data["sec"] = $sec;
    $data["usec"] = $usec;
    /* Create an array to store the data.
     * This is where you would generate the data
     * you wish to cache, such as making database
     * calls. */
    $ser_data = serialize($data);
```

```

    /* serialize the array for return */

    return ($ser_data);
}

$time = time(); /* get current timestamp */
$micro = microtime();
/* get current timestamp
   including microseconds */
list ($usec, $sec) = explode(" ", $micro);
/* Print the real current time */
print "<p>Current Time: $time, $sec, $usec</p>";
$cached_string = output_cache_fetch("Example: Fetch", "get_content($time, $sec,
$usec);", 30);
/* Call the function via the 'output_cache_fetch()'.

   If the content key exists and the lifetime hasn't
   expired, the function execution will be skipped
   and the cached data will be returned via the cache
   API call. */

$data = unserialize ($cached_string);
/* unserialize the data */
/* Print the cached time */
print "<p>Cached Time: " . $data["time"] . "," . $data["sec"]. "," . $data["usec"] .
"</p>";
print "<p><b>Refresh to see caching in action!</b></p>";
?>

```

The strength of *output_cache_fetch* is that it allows the developer to offload repetitious database calls by wrapping these calls in a function. The following example illustrates how this would work.

```

<?php
/* Note that this code is kept as simple as possible,
   with no return type checks etc., in order to focus
   on the caching features. */

/* Display greetings and read user info from database -
   this part must remain dynamic */

$user_id=($_SESSION['user_id']);
$result = mysql_query("SELECT name,country,airport
                      FROM users
                      WHERE id=$user_id");
list($name,$country_id)=mysql_fetch_array($result,MYSQL_ASSOC);
echo "<P>Welcome $name ".date("F j, Y, g:i a") . "</P> ";

```

```
/* Note that the code to be cached should be wrapped as
   a single function call (see argument 2 in above
   example) - this is to improve readability and code
   reuse (the code we want to cache is usually longer
   than one line.) */

// Display list of destination countries
$sql = "SELECT id,name FROM countries";
echo "<P>Destinations: ";
$destination_str=output_cache_fetch("destinations","GetQuery('$sql')",3600);
$destination_arr = unserialize($destination_str);
$out = "<P>Destination Airport: <SELECT NAME=\"destination_airport\"> ";
foreach ($destination_arr as $destination) {
    $out .= "<OPTION VALUE=\""
        . $destination['id'] ."\">"
        . $destination['name'] ."</OPTION> ";
}
$out .= "</SELECT></P> ";
echo $out;
echo "</P>";
/* In the first caching example (above), we cache the
   list of destinations. This is the same for every
   user, and so the cache id is a simple string. In
   the second example (below), the list of airports
   depends on the user's country. So, the country_id
   is added in the ID string. This will create a
   different cache copy for each continent. */
// Display list of airports in the user's home country
$sql = "SELECT id,name FROM airports WHERE country='$country_id'";
$airports_str = output_cache_fetch("airports_$country_id","GetQuery('$sql')",3600);
$airports_arr = unserialize($airports_str);
$out = "<P>Departing Airport: <SELECT NAME=\"depart_airport\"> ";
foreach ($airports_arr as $airport) {
    $out .= "<OPTION VALUE=\"".$airport['id'] ."\">".$airport['name'] ."</OPTION> ";
}
$out .= "</SELECT></P> ";
echo $out;
/* This function is more general purpose, meant to be
   used with output_cache_fetch () it performs an SQL
   query and returns the results as a serialized string.*/
function GetQuery ($sql_query) {
```

```

$result = mysql_query($sql_query);
$res_arr = mysql_fetch_array($result, MYSQL_ASSOC);
$res_str = serialize($res_arr);
return $res_str;
}
?>

```

Usage Notes:

You can use *output_cache_fetch()* to cache non-string types (e.g. arrays and objects) of PHP variables by using PHP's *serialize()* and *un-serialize()* to convert them to strings and vice versa.

output_cache_fetch requires the code which generates the cache data to be wrapped in a function. This allows the cache routine to skip the execution of this code if the data is already cached

output_cache_put() and *output_cache_get()* provide a direct way to store and retrieve data from the cache.

Prototype:

```

void output_cache_put(string key, mixed data)
void output_cache_get(string key, int lifetime)

```

***output_cache_put()* takes two arguments:**

1. The key value with which this data will be cached.
2. The data to be cached. (scalar, string, or serialized data).

output_cache_put() has no return value.

***output_cache_get()* takes two arguments:**

1. The key value with which the data is stored.
2. The lifetime that this data should be considered valid.

output_cache_get() returns the cached data, if it exists and is valid. Otherwise it returns false.

Usage Example

```

<?
if(($result = output_cache_get("TestFunctionResult", 30)) === false) {
    $result = microtime(); /* Current timestamp */
    print "<br><i>Fetching Fresh Content</i><br>";
    /* Should only print this every 30 seconds when
       the content is fetched fresh */
    output_cache_put("TestFunctionResult", "Cached: $result");
}
print "<b>$result</b><br>";
?>

```

Usage Notes:

The put/get routines are a simpler method for caching data than output_cache_fetch. They are typically used for the storage and retrieval of small bits of data.

Action Based Partial Page Caching

Action Based Partial Page Caching pertains to caching part of an output based on the occurrence of an action. This type of Caching is necessary in instances where it is preferable to refresh the Cache when an action occurs rather than time based.

For example: If we have a list of people who are "Currently Online" and we were to use time based caching, we would have to set an extremely short time limit to make sure that the list is updated at all times. We would also waste valuable system resources every time we refresh the cache. Instead, we can adopt a more efficient approach: refreshing the cache based on an action, for instance, every time a member goes online or logs out.

How do we do this?

There are two "Partial Page Cache" options based on an action:

1. Conditional Partial Page Caching
2. Button Based Partial Page Caching

Conditional Partial Page Caching

With this option, we predetermine conditions for caching and invalidating (If X occurs then do Y).

For example: we can cache our list of people who are "Currently Online" based on their log-on action. Whenever someone logs-on, his or her name will be added to the cache. Subsequently, when the same person logs-off we could set another condition will remove the name from the cache.

The following code example demonstrates how to empty the cache when a certain action occurs:

```
if (check_some_condition()) {  
    output_cache_remove_key (...);  
}
```

Button Based Partial Page Caching

With this option, we set a specific button to initiate refreshing the Cache (Pressing button X does Y).

For example: we can cache our list of people who are "Currently Online" based on a specific button that the person logging-in will press (such as: login, next, go, etc). Whenever someone presses the button, his or her name will be added to the cache. Subsequently, when the same person presses a different button, his or her name will be removed from the cache.

The following example demonstrates the button triggered Partial Page Caching technique:

```
<form action="clean_cache.php">  
    <input type="submit" value="Clear Cache">  
</form>
```

The 'action' attribute points to the clean_cache.php script. Therefore, when the user submits the form, clean_cache.php is executed+.

The clean_cache.php file is the same as the one that we put in the Cron Job example (see next chapter): it clears the cache (with output_cache_remove()) and then builds it again (With fopen("http://..")) - So we get a real cache refresh.

This tutorial details Action Based Partial Page Caching, with conditional or button-oriented options.

Chapter 8 - Appendixes

Appendix A - Configuration Check List

This Check List details all the Zend Platform configuration tasks in chronological order. This list can be printed and used as an extra aid for setting-up Zend Platform.

1. Configure Clusters and Groups when working in a cluster environment to enable event aggregation over multiple servers.
Zend Central | Server Status | Manage Cluster or use the Zend Central Shortcut
Zend Central | Console | Manage Cluster
2. Event Triggers, to modify default settings to suit the new environment:
PHP Intelligence | Event Triggers
Or use the shortcut Zend Central | Console | Event Triggers
3. Configure Action Rules, to send Event Details data by e-mail or to a URL:
Zend Central | Console | Configure Action Rules
4. Configure Performance, to define initial performance settings for Code Acceleration, Dynamic Content Caching, File Compression and Download optimization:
Performance | Settings
5. Configure Virtual Hosts and fine tune performance setting per file:
Performance | File View
6. Setup integration with Zend Studio Server in Zend Core.
7. Establish a persistent connection with Zend Studio for Debugging Profiling and Editing code in Zend Core.
8. Configure PHP settings to customize the php.ini and zend.ini to your environment:
Use the Zend Central Shortcut: Zend Central | Console | Configure PHP Settings
9. Use the Clone Wizard to apply configurations from one node to other nodes:
Zend Central | Console | Clone Wizard
Alternatively use the Quick Clone buttons to apply specific performance and PHP Intelligence settings.
10. Define User and Group permissions
Zend Central | User Management

Appendix B - Performance Lifecycle Check List

This Check List details all the Zend Platform performance Lifecycle tasks in chronological order. This list can be printed and used as an extra aid for calibrating Zend Platform.

1. Benchmark Web application, to establish optimization-starting point:
Performance | Testing | Analyze Site – Run Performance Tool
2. Calibrate Event rules to configure PHP Intelligence events to the Web application's performance parameters:
PHP Intelligence | Event Triggers
Or use the shortcut Zend Central | Console | Event Triggers
3. Benchmark Web application, to establish a second optimization-starting point:
Performance | Testing | Analyze Site – Run Performance Tool
4. Analyze Event Details, to pinpoint performance issues:
PHP Intelligence | Event List
Recommended: Focus on the following performance related event types:
 - Slow Script Execution (Absolute and Relative)
 - Slow Query Execution
 - Slow Function Execution
 - Excess Memory Usage (Absolute and Relative)
5. Apply Caching to boost Web application performance:
 - Define Dynamic Content Caching: Performance | File View (or from the Site Analysis results).
 - Apply Partial Page Content Caching APIs (see Tutorial)
6. Configure Acceleration to save code compilation time:
 - Acceleration Settings: Performance | Settings
 - Acceleration Blacklist: Performance | File View
7. Configure Compression to consume less bandwidth:
 - Compression Settings: Performance | Settings
 - Compression Blacklist: Performance | File View
Important: Deactivate compression entirely if the server is set to handle compression (Performance | Settings | File Compression).
8. Configure Optimization optimize script and detect encoded files:
Zend Central | Console | Configure PHP Settings | Zend | Zend Optimizer
9. Benchmark Web application, to view optimization boost:
Performance | Testing | Analyze Site – Run Performance Tool

Appendix C - Event Aggregation Mechanism

Introduction

This appendix covers the event aggregation mechanism in the Central Server. It will try to answer the fundamental question: "When are two events considered to be of the same origin (or cause) and therefore reported as one?"

Event properties

To answer this question we first have to define the different properties (or attributes) that define an event. Here is a list of the attributes that are used for aggregation along with a short definition:

- Event type - the type of the error that triggered the event (PHP error, Function error etc'). Perhaps the most important property since it also determines which other properties will be compared.
- Source file, Line number - the name of the PHP file and the line that contains the code that triggered the event. This file may not be the file that the user requested. Not all events have code location - e.g., "slow script" events and other events related to the whole script do not.
- Function name - the name of the function that contains the code that triggered the event. If the event happened in the global scope it's reported in the 'main' function.
- Location - one of two: either the server id of the server that triggered the event or the group id if the server belonging to an aggregated group.
- Aggregation Hint - this is a string that is supplied by the user to differentiate between pages that have the same URL but different parameters. If the user did not supply a hint the default hint is an empty string (The limit for Aggregation hints is 255 chars, longer hints will not be aggregated).
- Error text - the error text that was attached to the event.
- Script id - refers to the record for the script that the user requested (i.e., derived from original request URL).
- Severity - the severity of the event - currently, has two levels - regular and severe.

Another property that is taken into account is the event status. Only events that are not closed are aggregated.

Events are not aggregated when they are one of the following:

- Events of different types.
- Events that happened on different non-aggregated servers.
- Events with different aggregation hints.
- Events with different severity.

Zend Error Events

The following properties must be equal for events that are of type "zenderror":

1. Type (note: this is a Zend error type, like E_WARNING, not monitor error type)
2. Source file
3. Line number
4. Function name
5. Location

6. Aggregation hint

The Error text attribute must be 75% similar. (To learn more about text similarity read <http://uk.php.net/manual/en/function.similar-text.php>)

Function Error Events

The following properties must be equal for events that are of type "funcerror" or "dberror":

1. Source file
2. Line number
3. Function name
4. Location
5. Aggregation hint

If one of the events has an Error text attribute than the Error texts must be the same (not similar!).

Long Function Events

The following properties must be equal for events that are of type "longfunction" or "longquery":

1. Script id
2. Source file
3. Line number
4. Function name
5. Location
6. Aggregation hint
7. Severity

Custom Events

The following properties must be equal for custom events:

1. Type (this is the first parameter user provides)
2. Severity
3. Event text
4. Source file
5. Line number

Additional Events

The rest of the events are aggregated according to following attributes if two conditions are met:

1. The event type is one of the following: "devmem", "memsize", "devscript", "outsize" or "longscript".
2. The event has a script id attribute

For these events the following attributes must be equal:

1. Type

2. Script id
3. Location
4. Severity

Appendix D - Zend Platform Support

Zend Platform Support

Zend Platform Support provides Zend Product owners and prospective Zend Product owners with information regarding: System Requirements, Installation Instructions, General FAQ, Quick Start Guide and much more.

Visit: http://www zend com/support/support_platform.php

Zend Support Center

The Zend Support Center is your online destination for information and assistance for Zend's best-of-breed PHP products and technologies:

- Knowledge Base
- Support FAQ
- Submit a Support Ticket

Visit: <https://www zend com/support/index php>

Appendix E - zend.ini Configuration Settings

The following table lists the zend.ini directives. Directives marked with YES do not require restarting the server apply changes.

Accelerator Directives

Directive	Reload
"zend_accelerator.max_wasted_percentage"	YES
"zend_accelerator.max_warmup_hits"	YES
"zend_accelerator.consistency_checks"	YES
"zend_accelerator.force_restart_timeout"	YES
"zend_accelerator.perform_timings"	YES
"zend_accelerator.validate_timestamps"	YES
"zend_accelerator.max_cached_filesize"	YES
"zend_accelerator.revalidate_freq"	YES
"zend_accelerator.min_free_disk"	YES
"zend_accelerator.php_extensions"	NO*
"zend_accelerator.user_blacklist_filename"	NO*
"zend_accelerator.compress_blacklist_filename"	NO*
"zend_accelerator.compression"	NO
"zend_accelerator.compress_all"	NO
"zend_accelerator.enabled"	NO
"zend_accelerator.output_cache_enabled"	NO
"zend_accelerator.max_accelerated_files"	NO
"zend_accelerator.mmap_base_file"	NO
"zend_accelerator.httpd_uid"	NO
"zend_accelerator.memory_consumption"	NO
"zend_accelerator.allow_noshm"	NO

"zend_accelerator.output_cache_config"	NO
"zend_accelerator.output_cache_dir"	NO
"zend_accelerator.use_cwd"	NO
"zend_accelerator.preferred_memory_model"	NO
"zend_accelerator.dups_fix"	YES

Monitor Directives

Directive	Reload
"zend_monitor.max_var_len"	YES
"zend_monitor.warmup_requests"	YES
"zend_monitor.load_sample_freq"	YES
"zend_monitor.rotate_freq"	YES
"zend_monitor.reconnect_timeout"	YES
"zend_monitor.watch_functions"	NO*
"zend_monitor.watch_results"	NO*
"zend_monitor.collector_host"	NO
"zend_monitor.collector_port"	NO
"zend_monitor.log_dir"	NO
"zend_monitor.server_key"	NO
"zend_monitor.server_cert"	NO
"zend_monitor.collector_cert"	NO
"zend_monitor.enable"	YES
"zend_monitor.error_level"	YES
"zend_monitor.error_level.severe"	YES
"zend_monitor.silence_level"	YES
"zend_monitor.max_script_runtime_load_cutoff"	YES

"zend_monitor.report_variables_data"	YES
"zend_monitor.max_script_runtime"	YES
"zend_monitor.max_function_runtime"	YES
"zend_monitor.max_memory_usage"	YES
"zend_monitor.max_load"	YES
"zend_monitor.max_script_runtime.severe"	YES
"zend_monitor.max_function_runtime.severe"	YES
"zend_monitor.max_memory_usage.severe"	YES
"zend_monitor.max_load.severe"	YES
"zend_monitor.max_time_dev"	YES
"zend_monitor.max_output_dev"	YES
"zend_monitor.max_mem_dev"	YES
"zend_monitor.max_time_dev.severe"	YES
"zend_monitor.max_output_dev.severe"	YES
"zend_monitor.max_mem_dev.severe"	YES
"zend_monitor.mem_threshold"	YES
"zend_monitor.time_threshold"	YES
"zend_monitor.output_threshold"	YES
"zend_monitor.event_overload_threshold"	YES
"zend_monitor.disable_script_runtime_after_function_runtime"	YES
"zend_monitor.tmp_dir"	NO
"zend_monitor.longscript.enable"	YES
"zend_monitor.longscript.enable"	YES
"zend_monitor.longfunction.enable"	YES
"zend_monitor.zenderror.enable"	YES

"zend_monitor.devscript.enable"	YES
"zend_monitor.funcerror.enable"	YES
"zend_monitor.devmem.enable"	YES
"zend_monitor.outsize.enable"	YES
"zend_monitor.memsize.enable"	YES
"zend_monitor.load.enable"	YES
"zend_monitor.custom.enable"	YES

Debugger Directives

Directive	Reload
"zend_debugger.allow_hosts"	YES
"zend_debugger.deny_hosts"	YES
"zend_debugger.allow_tunnel"	YES
"zend_debugger.expose_remotely"	YES
"zend_debugger.network_trace"	NO
"zend_debugger.max_msg_size"	YES
"zend_debugger.httpd_uid"	NO

ZDS Directives

Directive	Reload
"zds.enable"	YES
"zds.mime_types_file"	NO*
"zds.log_file"	NO
"zds.min_file_size"	YES
"zds.disable_byterange"	YES
"zds.mmap_chunk"	NO
"zds.nice"	NO
"zds.allow_assert"	NO
"zds.child_max"	NO
"zds.poll_delay"	NO
"zds.uid"	NO

(*) Not reloaded now – maybe in future versions)

Chapter 9 - Index

A

Absolute 21, 24
accelerated 35, 45
acceleration 35, 47
Acceleration Blacklist 47
Acceleration Blacklist Files 35
Action 29
Action Rule parameters 29
Action Types 29
Actions Rules 29
add servers 12
Add/Edit Users 55
Adding a Server 12
Additional Rules 21, 22, 26
aggregate Events 12
Aggregate Hint 72
Aggregate Hints 82
Aggregate servers 66
Aggregation 28
Aggregation Rules 72
Alert Rules 16
Alert Window 68
Alerts 68
aliases 12
All66
All Files 35, 45
ALLGET 41
allowed 54
ALLSESSION 41
Always Report Errors 22
Analyze Site 90
Apache 26
API 28, 82
Array 41
audit trail 72
average 21

B

Backtrace 72
bandwidth 101
Blacklist 45, 47
Blacklisted 35
bottlenecks 23
Bridge 107

C

Cache 35
Cache Cleaner 41
Cache Lifetime 41
Cached 35, 45
cached files 35, 45

Caching 41
caching conditions 35, 41
Cancel 35
Central Server 12
Clean 35
Clone 53
Clone Configurations 12, 53
Clone Settings 53
Clone Wizard 53
Close Event 72
Close Selected 68
Clusters 12
compressed 35, 45
compression 35, 45, 47
Compression Blacklist 47
Compression Blacklist Files 35
Compression Test 90
Compression Test results 90
concurrent jobs 101
Concurrent Requests 101
configurations 35
Configure Alert Rules 16
Confirm Password 55
Connection errors 25
Content 41
Cookie 41
CPU 21
CPU Load 72
create a New Group 57
critical situations 27
Current Blacklisted Files 47
Current Existing Users 54
Custom events 28, 72

D

daemon 107
Database 26, 66
database cleanup 68
Database Error 25
database function errors 25
Database selection errors 25
databases 25
Deactivate compression 45
Debug URL 72
Debugger 72
Default Cache Lifetime 41
default configurations 16
Define Action Types 29
Define Alert Rules 16
Define Cache 35
Define Caching Settings 35

Delete Event	72
Delete Selected.....	68
diagnostic	72
Disable an Action Rule	29
Do not Cache.....	35
document root	35
Download.....	101
Download Server.....	101
Download tab	101
Download Test.....	101
Dynamic	41
Dynamic Caching Enabled	41
Dynamic Content Caching.....	41, 45
E	
edit	54
Editing a User	55
Editing Groups.....	57
Editing Users	55
e-mail	29, 72
Email Report	29
End User.....	66
Environment.....	107
Error Description.....	72
error-reporting criteria.....	22
Errors.....	22
Event	21, 22, 23, 24, 26, 27
Event Data.....	72
Event List.....	68
Event Name	66
Event Occurrence Info	72
Event Report	29, 72
Event Reports.....	29, 72
Event Summary	66
Event Summary table	66
Event Type.....	16, 68
Event Types	16
events	25, 27, 66, 68, 82
Events From.....	16, 68
Excess Memory.....	24
Excess Memory Usage	24
F	
File Compression.....	45
File Settings	35
File View	35
Filtering	66, 68
Filtering Alert Rules	16
Filtering Table Data	66
Fine Tune Caching Conditions	35
Full Page Content Caching	41
function	26
Function Data.....	72
function errors	22, 25
Function Execution.....	23
Function Usage.....	28
functions	22
G	
General database function errors	25
Get.....	41
Global Events.....	82
global variable	82
Group	54, 66
Group Name	54
groups	12
GUI Directory.....	12
gzip.....	45
H	
Handle Groups	54
Hard errors.....	22
health	27
Host	66
HTTP Server	35
I	
Ignore Event.....	72
Ignore Selected.....	68
Included Files.....	72
Inconsistent Output Size.....	26
Integration	72
investigating	72
J	
Java Bridge.....	107
Java Environment	107
Java Home	107
Java Status.....	107
Java Vendor.....	107
Java Version	107
L	
Last Download	101
Load Average.....	27
Logged in as	54
login	54
M	
Manage Servers/Groups	12
management	54
Max Clients.....	101
Maximum Cache Size	41
Maximum Cached File Size	41
Memory	24
Minimum Free Disk space	41

Monitor.....	16, 27	Remove	12, 54
monitor_set_aggregation_hint	82	Reopen Event.....	72
multiple users.....	54	Reopen Selected.....	68
N		REQUEST	41
N/A Test Results	90	Requested URL.....	72
New Group.....	57	requests.....	21
New Users.....	55	resolving events	72
Node Installation.....	12	Restart.....	107
None.....	45	Restore Defaults.....	35
Notices	22	rule-based notification.....	29
O		Rules	16, 29
Only Cached Files.....	45	S	
operator	22	Save.....	35
OS Name	107	scripts.....	24, 28
OS Version.....	107	Send a report via e-mail	29
output site	66	sending the information.....	29
P		Server	41, 66
parameters	21	Server by name.....	66
Partial Page Content Caching	41	Server Name.....	66
Password	55	servers	12
Performance.....	35, 90	SESSION.....	41
performance configurations	35	severe event	22
Performance Lifecycle	16	Severity	68
Performance Monitoring Event.....	21, 23, 24	Severity Level	72
Performance Test	90	Show Source Code.....	72
Performance Test results	90	silence operator.....	22
Permission Group	54	silenced	22
permissions.....	55, 57	simulation	101
Permissions Group	55	Site Analysis	90
PHP	66	Site Analysis Report	90
PHP Error	22	slow function	21
PHP events.....	66	Slow Function Execution	23
PHP function.....	23	Slow Query	26
Port	12	Slow Query Execution	23, 26
Preserve Event	72	Slow Script Execution	21
Preserve Selected.....	68	source code	72
Previous Test Results	90	Source Info.....	72
processes.....	21, 27	SSL	12
Production.....	22	Start.....	107
profile	72	Statistics	107
Profile URL	72	Status.....	68, 72, 107
Q		Stop	107
QA.....	22	Submit a report to a Specified URL.....	29
Query Execution	26	supported databases	26
R		Suppress	21
Refresh.....	107	T	
Relative	21, 24	table.....	66
Relative Events	24, 26	Test	101
		Test Download	101
		Test Report.....	101
		Test Results.....	90, 101

Test URL.....	72	View Tree By field.....	53
Testing	90, 101	Virtual Host	68
Time Filter	68	Virtual Hosts	35
Title	72	Virtual Hosts List	35
Tree View	35	W	
U		Warnings.....	22
Undo	35	Watched Functions.....	22
Ungrouped	66	X	
un-zipped	45	XML.....	72
Update Blacklists.....	35	XML data.....	29
Update the Blacklist.....	47	XML Report.....	29
URL.....	29, 101	Z	
User	54	ZDS.....	101
User Management	54, 57	Zend Error.....	72
User Name	55	Zend Monitor	16
users.....	55	Zend Studio.....	72
V			
Variables	41, 72		
vhost.....	72		